Escuela Politécnica Superior

Aplicación Web: Recetarium

.do en Ingeniería Informática

Trabajo Fin de Grado

Autor:
José Vicente Orts Romero

Tutor/es:
Domingo Gallardo López

Universitat d'Alacant
Universidad de Alicante

Julio 2016

CONTEXTO

Esta idea nace con la intención acercar el mundo de la cocina a los que, como yo, desconocemos pero admiramos a partes iguales.

Aprender a cocinar es casi imprescindible hoy en día para casi cualquiera, y la mayoría solo hacemos una o dos recetas que nos enseñaron nuestros familiares, amigos o porque la hemos visto en alguna revista, programa de televisión o en algún post en internet.

Actualmente, con el auge de los programas de cocina, como Master Chef o Top Chef en España, cada vez más gente está empezando a tener interés por nuevas recetas, pero casi siempre se nos olvidan al par de días y nunca te acuerdas ni del nombre.

Recetarium (el nombre de este proyecto) intenta solventar esto. ¿Como? Muy simple, *Recetarium* se trata de una aplicación creada para suplir esos momentos en los que quieres buscar una receta en poco tiempo y poderla guardar en algún sitio. Aparte, también podrá aportar sus ideas de cocina al mundo, adjuntando fotos y videos, comentar acerca de alguna receta, conocer gente y comunicarse con ellos.

AGRADECIMIENTOS

Me gustaría tener unas palabras de agradecimiento hacia todas las personas que me han ayudado a llegar donde estoy.

En primer lugar a mis padres, José Vicente y Mº Dolores, por su apoyo incondicional, por convertirme en el hombre que soy hoy en día, por ser el mejor ejemplo a seguir y, sobretodo, hacerme sentir el hijo más afortunado del mundo.

A mi hermana pequeña, Bárbara, por su optimismo, vitalidad y por estar siempre ahí cuando no se la necesitaba.

A mis compañeros de carrera, que aun siendo varios e ir separándonos debido a las clases, siempre hemos estado juntos para darnos apoyo en los momentos difíciles, ayudándonos mutuamente y haciendo de la carrera algo que no quieres que acabe.

A mi tutor Domingo Gallardo López por aceptar mi proyecto, por su ayuda y consejos, pero más que nada por las clases divertidas que tuve con él, haciendo de la materia un juego.

A todos los docentes que he tenido, que además de conocimientos, me han despertado una curiosidad y una pasión de ir aprendiendo más acerca de esta profesión.

Y por último, a la Universidad de Alicante y la Escuela Superior Politécnica, a la que siempre estaré agradecido por prepararme para mi futuro laboral en una profesión que cada vez amo más.

ÍNDICE

ÍNDICE DE ILUSTRACIONES

1. INTRODUCCIÓN

En este proyecto se propone desarrollar una aplicación para ver, guardar y compartir recetas. En ella los usuarios pueden crear, editar y borrar recetas propias con total libertad, y ver y comentar en las recetas de todos los usuarios. También podrán marcar como favoritas y puntuar recetas. Y por último, los usuarios podrán contactar con otros usuarios haciendo amigos.

Esta aplicación posee tres partes. En el servidor esta la API REST desarrollada en Java con Play Framework usando el patrón MVC. Para guardar los datos se ha utilizado MySQL en desarrollo, base de datos en memoria para los test, y MySQL en producción (heroku). En el cliente Web se encuentra una aplicación en AngularJS (HTML5 + CSS3 + Javascript) conectada con la API del servidor siguiendo el mismo patrón MVC.

Para guardar todo el código del proyecto se ha utilizado Git como SCV (Sistema de control de versiones). Durante el desarrollo se va a seguir el flujo de trabajo de gitflow. En él, existen dos ramas principales: master (producción) y develop (desarrollo). En la rama master solo contiene commits que pueden subirse a producción, mientras que en develop están todos los commits, junto con todas las ramas auxiliares. Las ramas auxiliares pueden ser: feature, hotfix o release. En las ramas feature se procede a desarrollar nuevas implementaciones o soluciones de bugs. En las ramas hotfix solo se arreglaran fallos graves que pueden ocurrir tanto en producción como en develop. Estas ramas son fusionadas a todas las ramas que se estén usando. Y las ramas release se utilizan para preparar el código para producción realizando los últimos pequeños cambios antes de pasar a master.

Para hacer las pruebas en el servidor se ha utilizado el modelo de CI (Continuous Integration). Esta práctica se utiliza más en proyectos con múltiples miembros donde cada integrante del equipo de desarrollo integra su trabajo frecuentemente (mínimo una vez al día). Con cada integración se verifican que la aplicación siga pasando los tests. En mi caso, estos test se han realizado con la ayuda del framework de Play y la librería JUnit en la parte del API REST y Karma JS para la parte del cliente Web. Todo esto se llevará a cabo haciendo uso de una metodología que permita llevar un control exhaustivo del progreso del proyecto y garantizar su entrega libre de errores.

2. OBJETIVOS

Los objetivos principales son crear una aplicación API REST usando el framework de Java Play Framework 2.4.x y un cliente Web en AngularJS, usar un modelo de CI para integrar nuevos cambios a ambas aplicaciones, y desplegar las aplicaciones en la plataforma Heroku.

La idea de crear una aplicación basada en una API REST es el de poder separar el back-end (API REST) de la aplicación del front-end (Interfaz del usuario), pudiendo así crear múltiples front-ends usando la misma API con los mismos datos. Esto permite una flexibilidad a la hora de distribución de la aplicación, ya que solo se necesitan crear una nueva interfaz del usuario y asumir toda la lógica desde la API.

Para este proyecto se eligió este tipo de aplicación para desarrollar una interfaz en Angular con material y, en un futuro, crear una aplicación en Android que use los mismos datos sin tener que replicar la lógica.

Por otro lado, el objetivo principal desde el punto de vista del usuario es el de crear y compartir recetas con otras personas. A continuación se indican, de forma más detallada, las diferentes características que debe tener la aplicación.

- Permitir el registro e identificación de usuarios. Los datos se almacenaran de forma segura y se controlará que no se pueda acceder a partes privadas sin autorización. La autorización se realizará mediante JWT.

- Buscar recetas por etiquetas y texto.

- Ver detalladamente recetas. Dentro de ella se podrá añadir a favoritos, puntuar y comentar recetas y responder a otros comentarios.

- Crear, editar y borrar recetas. Se podrán configurar varios datos, entre ellos la visibilidad de la receta, la cual tendrá 3 posibilidades: pública, solo amigos y privada.

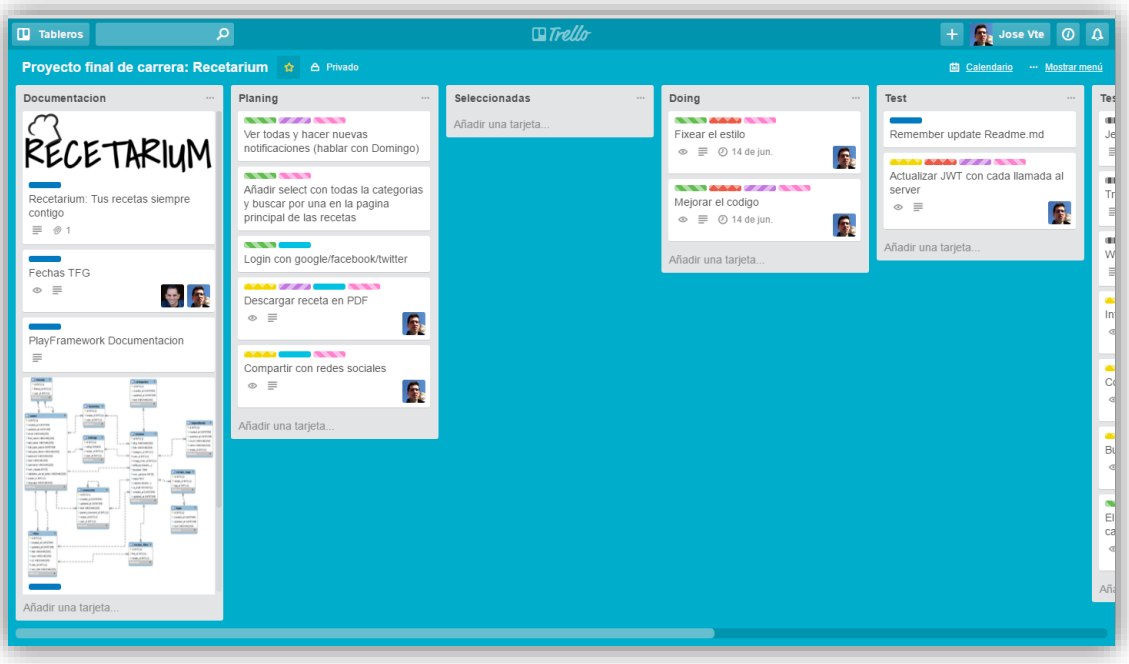
- La web estará adaptada tanto para PC como para móviles.

- Desplegar la web en Internet

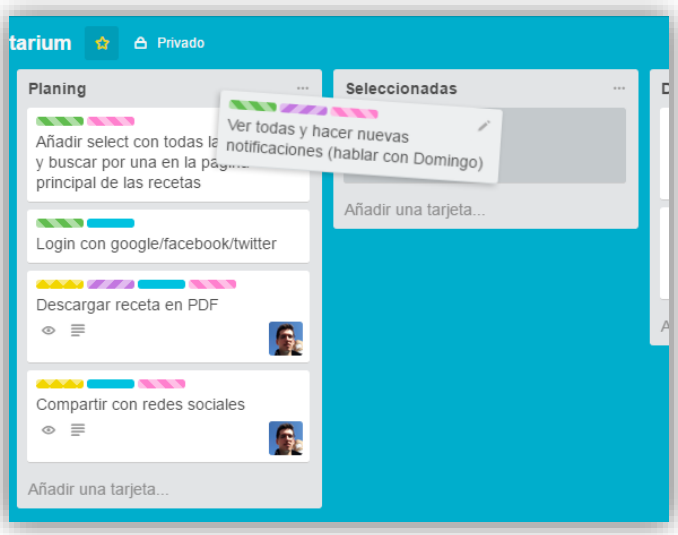
3. HERRAMIENTAS

3.1. Trello

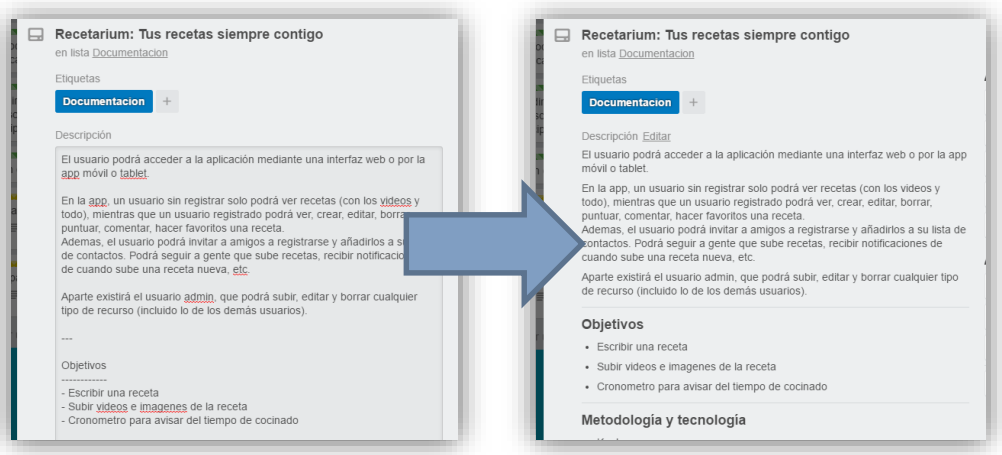
Trello es una aplicación web que permite la gestión de proyectos usando la metodología Kanban. Los proyectos están representados como *Tableros*, que contienen *listas*, las cuales corresponden las *listas de tareas* de Kanban.



Dentro de cada lista hay una serie de tarjetas o cards que se corresponden con las *tareas*. Estas tarjetas pueden ser movidas entre listas mostrando el progreso del proyecto.

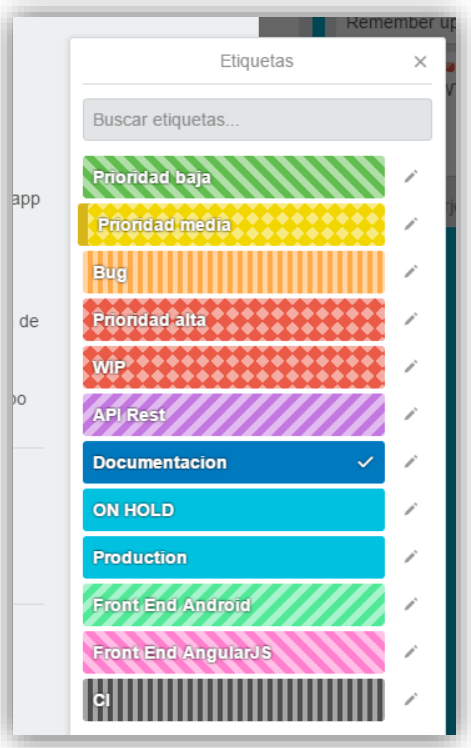


Cada tarjeta define una tarea a realizar, con un texto simple pero conciso. Pero si se necesitan dar más detalles, aportar enlaces de interés u otra información útil, las tarjetas también tienen un editor Markdown para añadir dicha información.

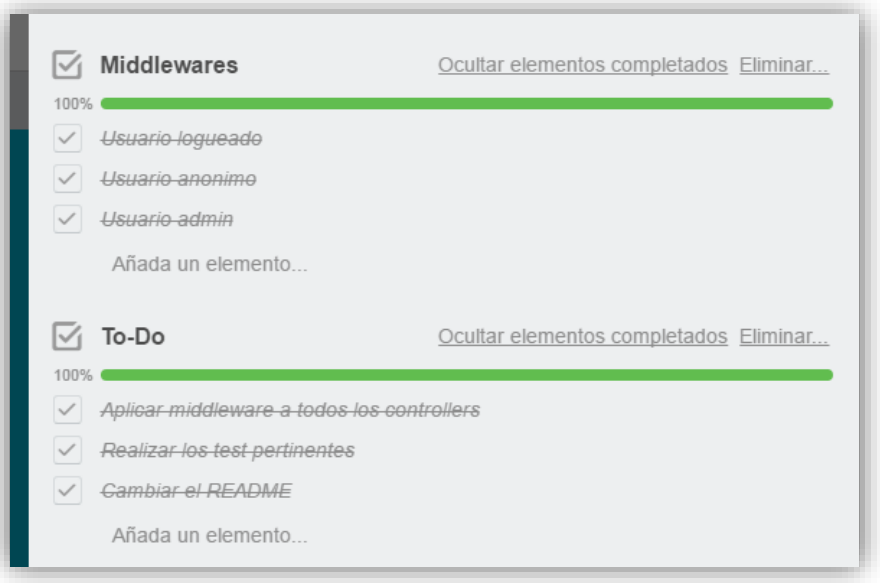


Aparte de la descripción, las tarjetas también pueden contener:

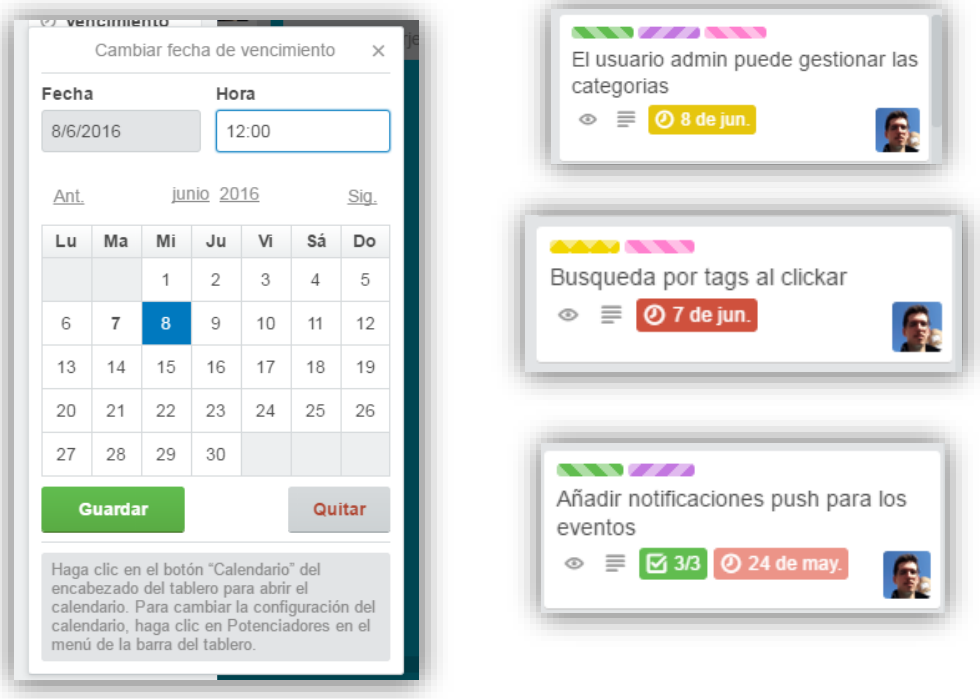
- Miembros: Generalmente son los usuarios que se encargaran de realizar la tarea.
- Etiquetas: Nombres y/o frases que permiten agrupar tarjetas. Ej.: Documentación, Hotfix, Prioridad media. Normalmente suelen estar acompañadas de un color que resalte el mensaje, como un rojo para las tarjetas con Prioridad alta:

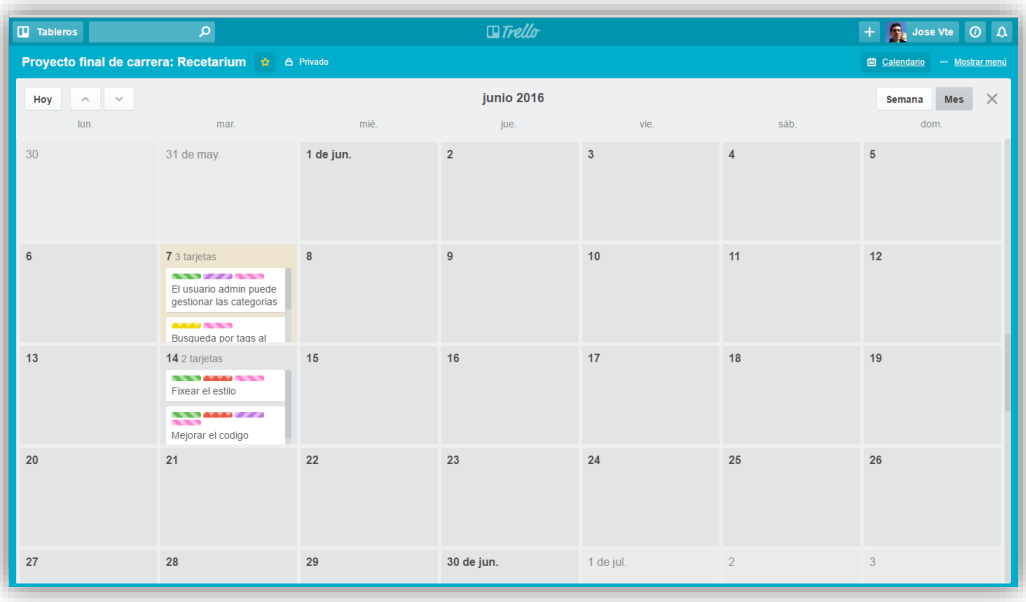


- Checklist: Como su propio nombre indica, una lista de subtareas que son tachadas una vez realizadas. Una tarjeta puede tener múltiples checklist:

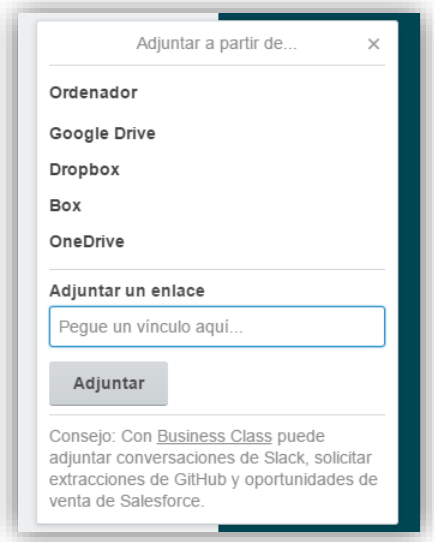


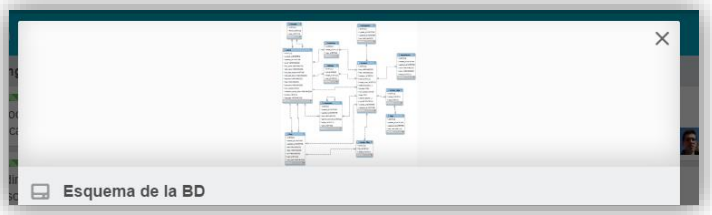
- Fecha de vencimiento: Se define una fecha límite para la tarjeta. El sistema notificará cuando la tarjeta esta próxima a terminar. También existe un calendario donde se pueden ver todas las tarjetas que tienen fecha de vencimiento, como si se tratara de Google Calendar:

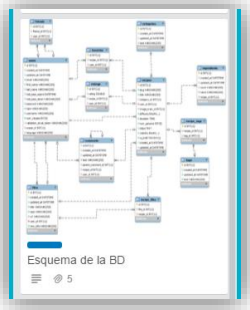




- Documento adjuntos: Se puede adjuntar cualquier tipo de archivo desde múltiples fuentes. Las imágenes pueden ser usadas como portada de las tarjetas:





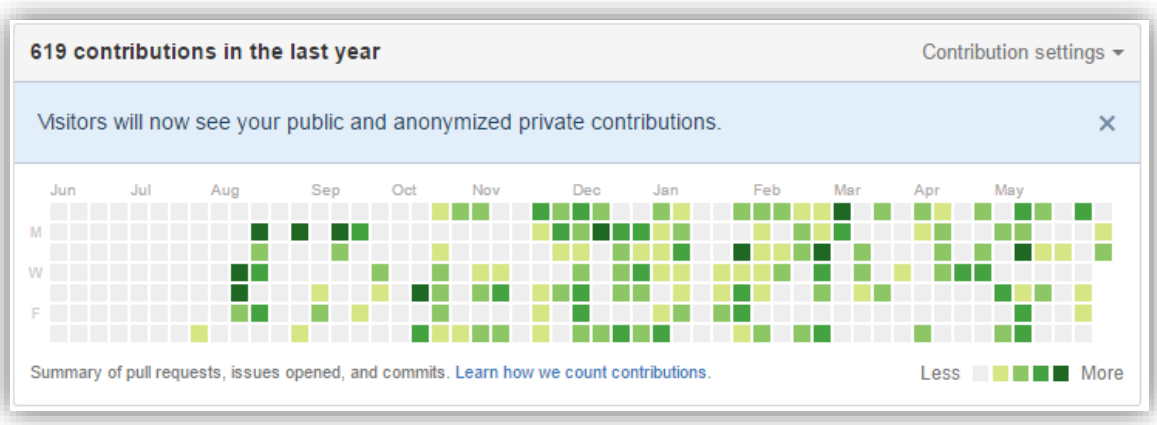


3.2. GitHub

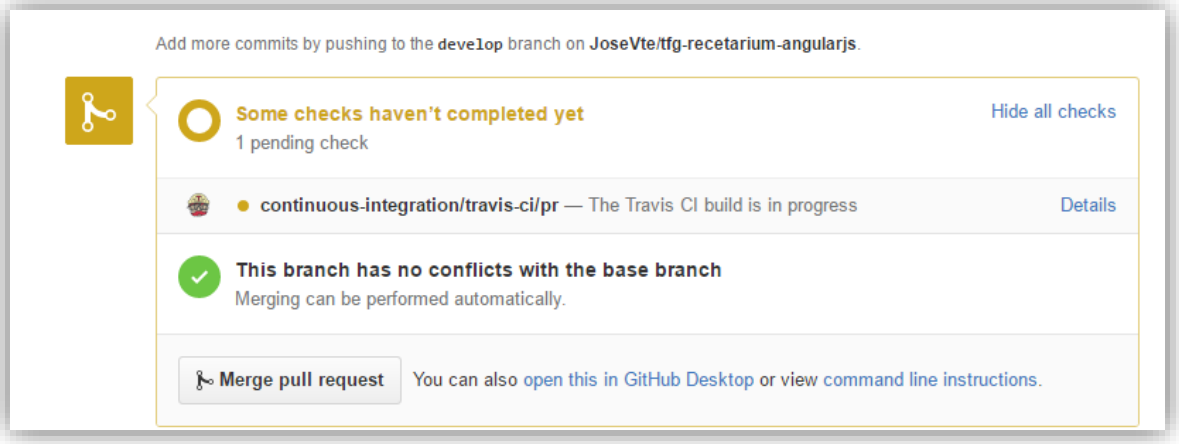
GitHub es una plataforma de desarrollo colaborativo, la cual permite alojar proyecto usando el sistema de control de versiones Git. Todo el código que se almacena es público salvo que se creen repositorios privados con una cuenta de pago o una cuenta de estudiante.

La principal ventaja de usar GitHub como SVC es la facilidad con la que se puede compartir código entre usuarios, comentar y aportar tu propio código mediante *Pull Requests* e *Issues* públicos y que gente opine sobre los cambios; y crear una Wiki y pagina web para tu código.

También te permite hacer un seguimiento de tu trabajo mediante graficas:



Otra característica que tiene GitHub, la cual comparte con otras plataformas, son los WebHooks. Un WebHook permite alterar el contenido de una página o aplicación web mediante *callbacks*. Algunos de los más usados son plataformas CI (Continuous Integration), como Travis o Jenkins:



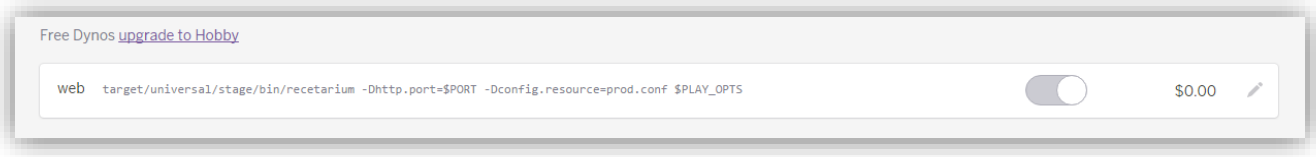
3.3. Heroku

Heroku es una PaaS (Platform as a Service) o Plataforma como servicio de *computación en la nube*, también conocido como *cloud computing*. En resumen, se trata de una plataforma que ofrece servicios de computacion a través de una red.

Heroku fue una de las primeras en ofrecer este tipo de servicio y soporta una amplia variedad de lenguajes, entre ellos Scala y Node.js.

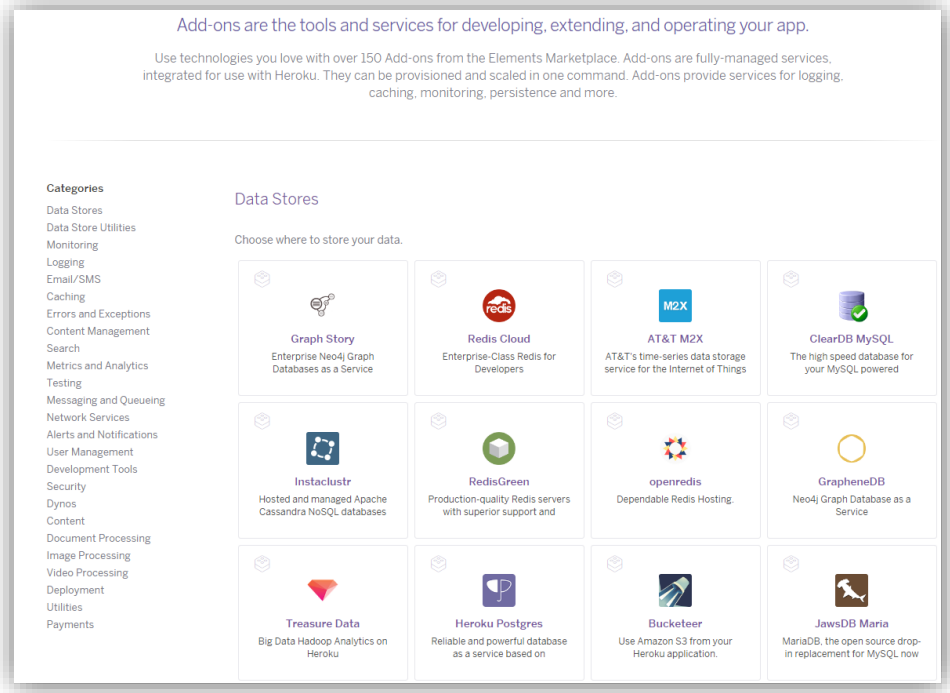
3.3.1. Dynos

Es la pieza fundamental de la arquitectura de Heroku. Los Dynos son unidades, basadas en contenedores Linux, que proveen el entorno idóneo para la ejecución de la aplicación. Cada Dyno está aislado del resto, por lo que no se pueden modificar archivos de otros Dynos.

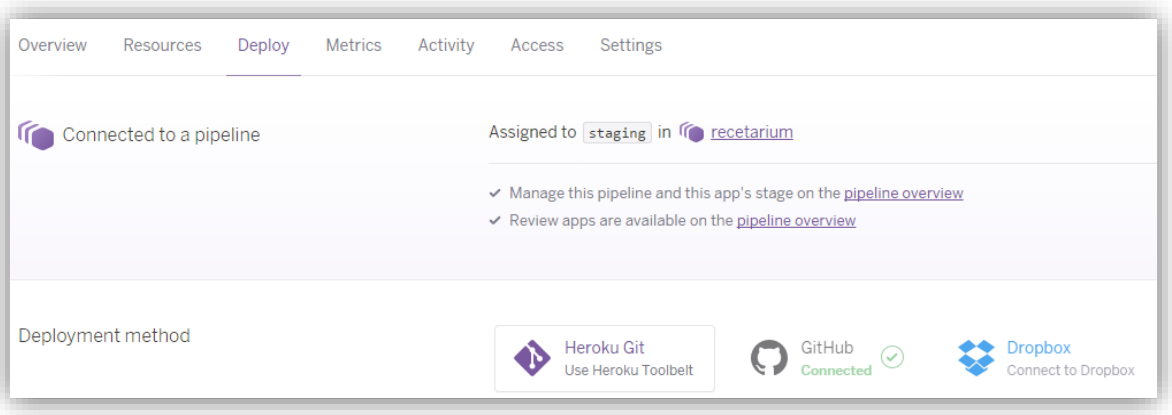


Los comandos que ejecuta Dynos están definidos en un archivo Procfile en la raíz de la aplicación. Los Dynos poseen una alta capacidad de crecimiento y elasticidad. Se pueden añadir y activar tantos Dynos como requiera la aplicación.

Los Dynos también son ampliables mediante *ADD-ONS*. Estos añaden nuevos servicios, como bases de datos o almacenamiento en disco.

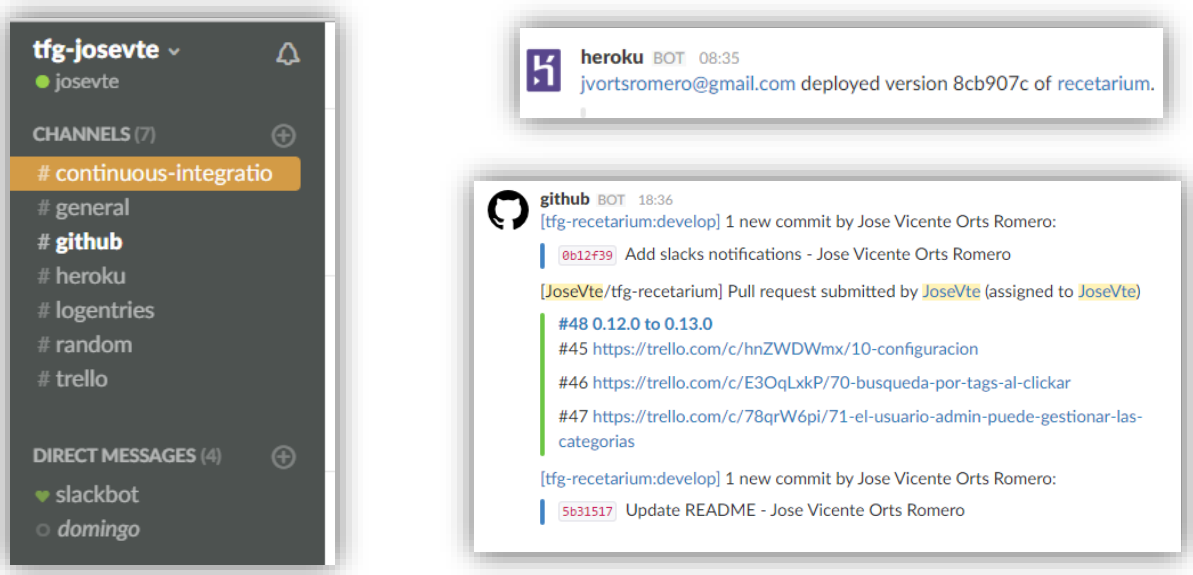


Otra característica es que permiten el despliegue de aplicación desde un repositorio alojado en GitHub, una carpeta de Dropbox o usando la aplicación en línea de comandos de Heroku.



3.4. Slack

Es una herramienta de comunicación en equipo que permite organizar salas de chats por temáticas, grupos privados y chat directo.



Slack integra una gran cantidad de servicios externos así como integraciones realizadas por la comunidad. Estos servicios deben estar asignados a un canal, y un canal puede tener múltiples servicios distintos. Algunos de los servicios que ofrece son:

- Heroku: este servicio ofrece información sobre despliegues de aplicaciones.
- GitHub: informa acerca de todo lo que ocurre en el repositorio, desde nuevos commits hasta Pull Requests aprovados.
- Trello: cualquier cambio y movimiento de tarjetas es notificado por este servicio.

- CI (Jenkins, Travis y Wercker): notifica si los test ejecutados en la plataforma han pasado exitosamente.

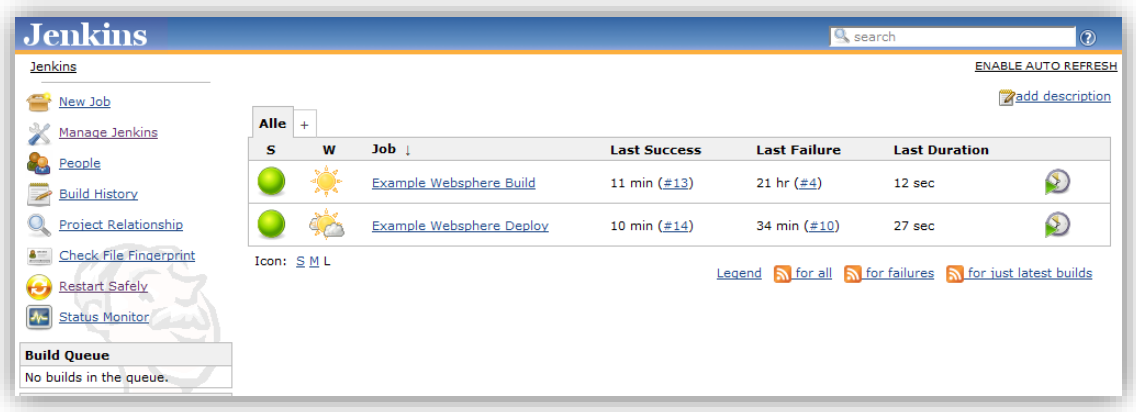
Todos los servicios pueden ser configurados para recibir solo ciertas alertas desde el panel de control de Slack.

3.5. Servicios de CI (Jenkins, Travis y Wercker)

Consiste en hacer *integraciones automáticas* de un proyecto lo más a menudo posible para así poder detectar fallos cuanto antes. El proceso habitual suele ser el de compilar y ejecutar los test cada cierto número de horas, pero también existe una variante que consiste en ejecutar los test por cada cambio subido al repositorio.

3.5.1. Jenkins

Una de las características de Jenkins es la facilidad a la hora de configurar y lanzar los test. Permite añadir plugin externos, añadir librerías y variables de entorno, etc. Para ello, Jenkins debe tener un servidor propio para ser instalado donde se ejecuta, ya sea cada cierto tiempo como cuando se le envía un WebHook.



3.5.2. Travis

A diferencia de Jenkins, en Travis no hace falta tener un servidor propio para ejecutar integraciones. Travis solo soporta proyectos en GitHub, tanto públicos (gratis) como privados (no gratis). El proyecto de Travis es open-source y está alojado en GitHub por si se quisiera implementar en un servidor privado.

Para configurar Travis, se debe crear un fichero *.travis.yml* en la raíz del proyecto. Aquí se definirán tanto el tipo de entorno como pasos (comandos) que se necesitan para pasar los test.

README.md .travis.yml

```
language: scala
sudo: false
jdk:
- oraclejdk8
scala:
- 2.10.4
- 2.11.2
sbt_args: -J-Xmx4g
script:
- sbt ++$TRAVIS_SCALA_VERSION test
cache:
  directories:
  - "$HOME/.ivy2/cache"
```

.travis.yml

```
language: node_js
node_js:
- '0.10'
- '0.12'
- '4'
before_install:
- export CHROME_BIN=chromium-browser
- export DISPLAY=:99.0
- sh -e /etc/init.d/xvfb start
before_script:
- npm install
- npm install -g jasmine-core
script:
- npm test
```

3.5.3. Wercker

Al igual que Travis, Wercker utiliza un fichero .yml de configuración alojado en la raíz de la aplicación con todos los pasos necesarios para poder ejecutar los pasos necesarios:

travis.yml wercker.yml

```
# This references a standard debian container from the
# Docker Hub https://registry.hub.docker.com/_/debian/
# Read more about containers on our dev center
# http://devcenter.wercker.com/docs/containers/index.html
box: dohque/docker-sbt
# You can also use services such as databases. Read more on our dev center:
# http://devcenter.wercker.com/docs/services/index.html
# services:
    # - postgres
    # http://devcenter.wercker.com/docs/services/postgresql.html

    # - mongodb
    # http://devcenter.wercker.com/docs/services/mongodb.html

# This is the build pipeline. Pipelines are the core of wercker
# Read more about pipelines on our dev center
# http://devcenter.wercker.com/docs/pipelines/index.html
build:
    # Steps make up the actions in your pipeline
    # Read more about steps on our dev center:
    # http://devcenter.wercker.com/docs/steps/index.html
  steps:
    - script:
        name: tests
        code: sbt test
```

Usa un contenedor Docker, el cual puedes cambiar y configurar para las necesidades de la aplicación.

Aquí se definen los pasos que se van a ejecutar para realizar los test.

3.5.4. Problemas con las plataformas de CI

Durante el desarrollo de la aplicación han ocurrido varios problemas relacionados con las plataformas de CI.

En Jenkins, al tener que ser un servidor propio, todos los servicios gratuitos tenían un límite de capacidad o uso de CPU o RAM, por lo que a la larga han causado problemas y se ha descartado como plataforma para esta aplicación, pero aun así es la mejor opción para equipos de desarrollo grandes.

En Travis ocurrió un fallo de escalabilidad, ya que el servicio que ofrece gratuito tiene limitación en cuestión de RAM, los test para la API REST usan más RAM de la que dispone y cuando la ocupan toda ocurre un error de Java y los test fallan.

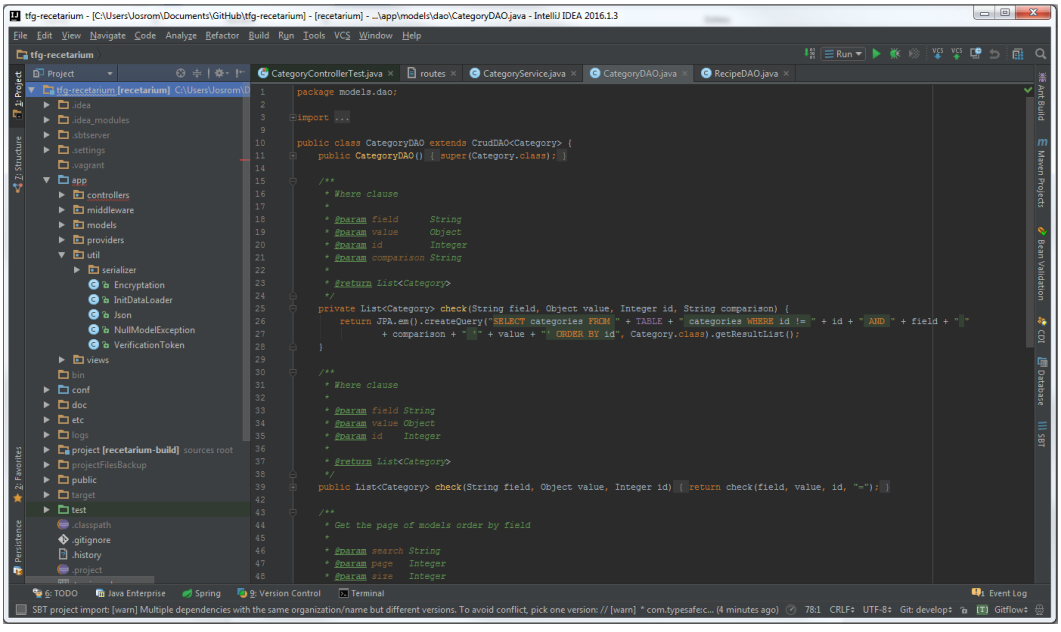
En Wercker el problema es que no se puede configurar el lanzamiento de los test, se ejecutan con cada commit subido al repositorio. No es un fallo grave pero sí que se ve reflejado cuando se sube a una rama secundaria un commit con el trabajo en WIP.

3.6. Editores de código

Durante el desarrollo del proyecto se han usado dos editores de código principalmente, uno para la API REST y otro para el front-end en AngularJS.

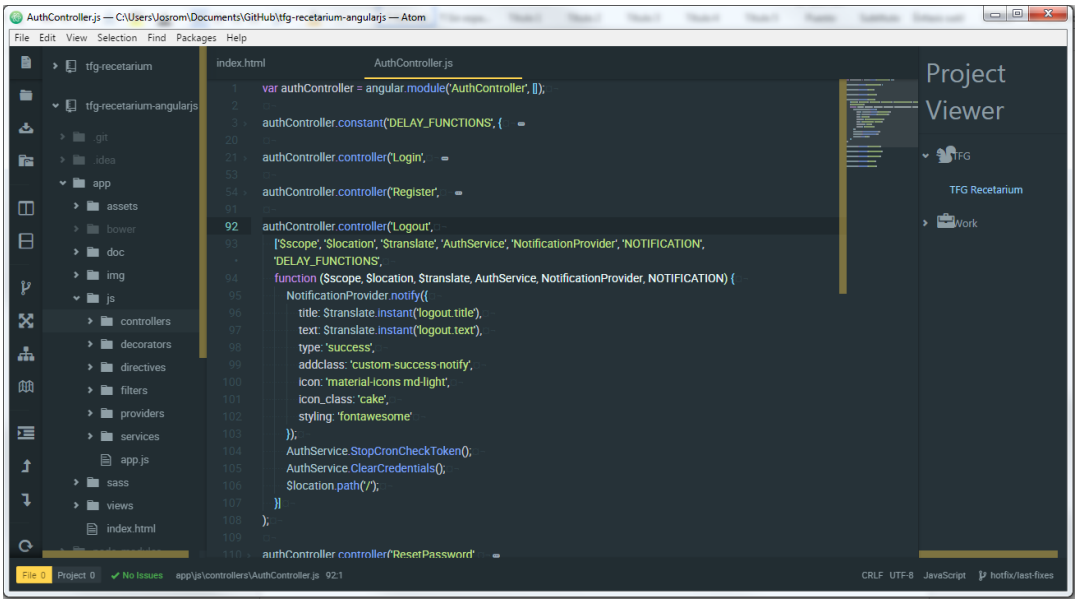
3.6.1. Intellij IDEA 2016

Desde las últimas versiones, Intellij se ha vuelto un IDE para Java potente, soportando sintaxis de frameworks como Play. Por ello se ha elegido este IDE para desarrollar toda la API REST, agilizando el trabajo y detectando los errores mucho antes que si tuviéramos que escribirlo en un editor normal.



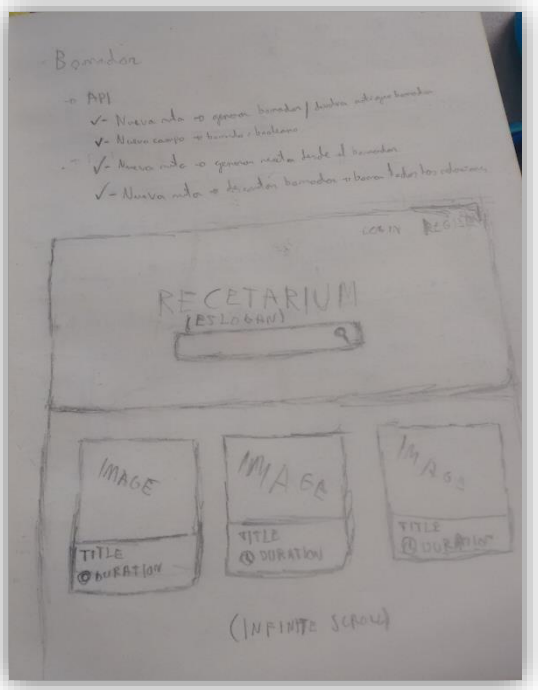
3.6.2. Atom

Atom es un editor de texto gratuito con una interfaz limpia, que permite añadir plugins, snippets y otras funcionalidades. Es muy parecido a SublimeText, otro editor de texto gratuito. No existe ninguna razón por la que para este proyecto se usara uno u otro.



3.7. Mockups

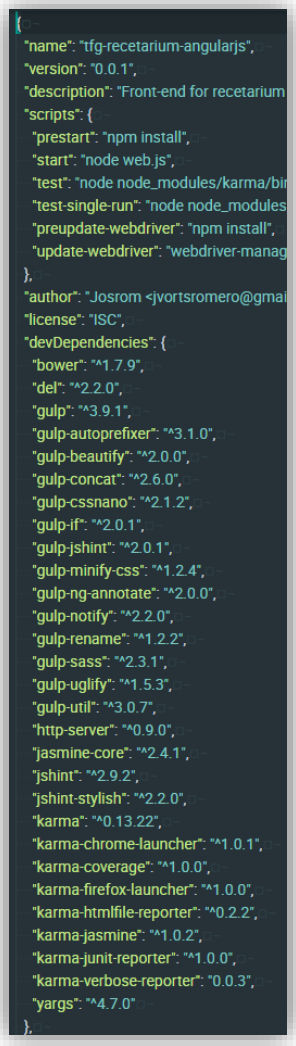
Para los mockups de la página web no se ha usado ninguna herramienta, si no que se han dibujado en una libreta con papel. Estos dibujos son llevamos a las reuniones de Scrum para poder ser discutidos y cambiados fácilmente. Aparte de los dibujos, en la libreta se apuntan también detalles y mejorar de la aplicación.

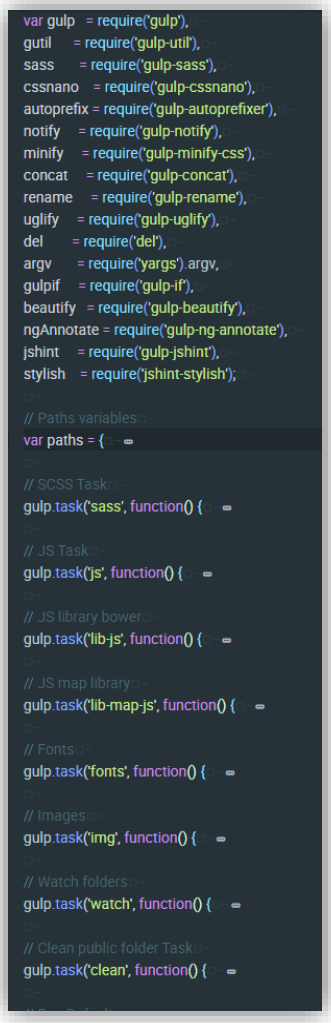


3.8. Bower, npm y gulp

Para el front-end se han usado muchos módulos de AngularJS, por eso se ha optado por una integración de las librerías usando gulp. La combinación de estos 3 programas permite el manejo de librerías JS y CSS de manera fácil y organizada. Basta con iniciar bower (bower init) y npm (npm init), e ir añadiendo las librerías necesarias, como AngularJS (bower install angularjs --save).

El resultado es tener 2 ficheros con todas las librerías necesarias para que la aplicación funcione y otro fichero para integrar todas las librerías: bower.json, package.json y gulpfile.js





4. TECNOLOGÍAS

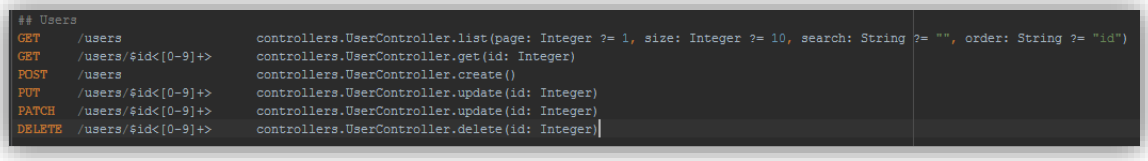
Las principales tecnologías usadas para el desarrollo de la aplicación han sido Play Framework como base de la API REST, JPA y MySQL como acceso y almacenamiento en base de datos y AngularJS como base del cliente Web.

4.1. Play Framework

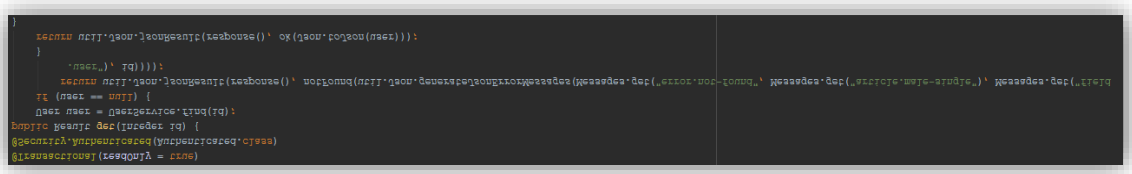
Play es un framework open source para aplicaciones Web escrito en Java y Scala el cual sigue un modelo arquitectural MVC (Modelo-Vista-Controlador). Al estar escrito en Scala comparte gran parte de las librerías con Java, por lo que el desarrollo puede ser tanto en Scala o Java y obtener el mismo resultado.

La aplicación de *Recetarium* está escrita en Java, usando como base la arquitectura API RESTful, donde se le han añadido algunos cambios para ciertas operaciones específicas, como subir múltiples imágenes.

En Play, para definir nuevas operaciones, empezamos por añadir dicha operación, con los parámetros que soporta:



A continuación, en el controlador propio de la operación, se añade el código para dicha operación, devolviendo un objeto tipo Result que contiene el estado de la operación y los datos en el formato que se le diga (JSON, XML, texto plano,…).



4.2. JPA y MySQL

JPA (Java Persistence API) es un framework para Java SE y Java EE. Permite interactuar con una base de datos usando un mapeo objeto-relacional, lo que hace posible usar objetos como instancias de la base de datos.

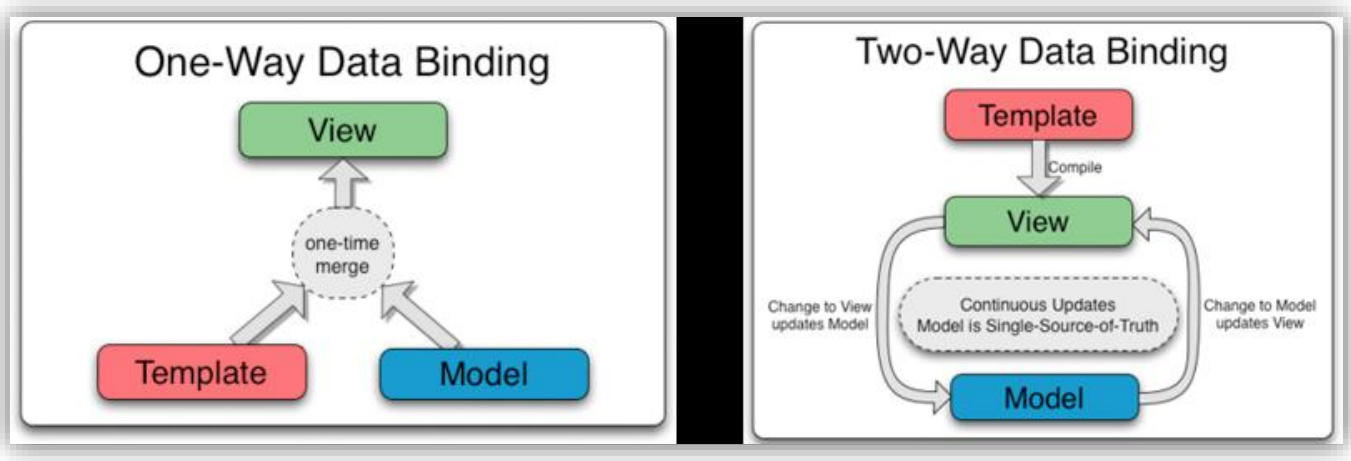
Usa JPQL (Java Persistence Query Language) como lenguaje de consulta orientado a objetos. Está inspirado SQL y la sintaxis es parecida, pero opera con objetos de JPA en lugar de tablas de la base de datos.

```
SELECT users FROM User users ORDER BY users.id
```

4.3. AngularJS

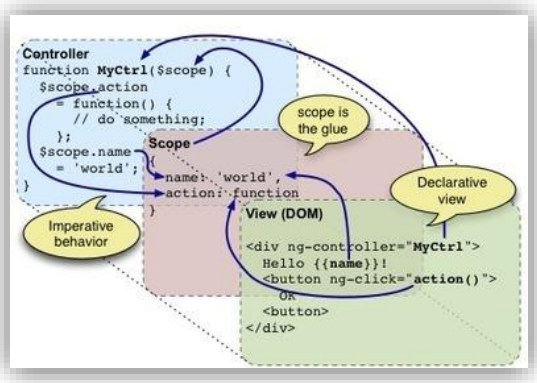
Es un framework MVC para Javascript para desarrollar aplicaciones SPA (Single-Page application), las cuales se identifican por cargar el contenido necesario para la página actual, y cargar dinámicamente el resto cuando se requiera, Web para ofrecer una experiencia fluida y más cercana a una aplicación de escritorio.

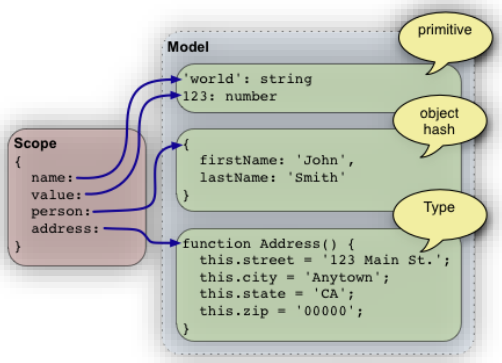
Una de las características principales es que no es necesario acceder al árbol DOM para obtener el valor de un elemento, usando las variables $scope y $rootScope. Esto se consigue usando una técnica llamada Two-Way data binding, cuando se cambia el modelo se actualiza la vista y viceversa:



4.3.1. Vocabulario

- **Scope**: Es el responsable de detectar los cambios en el modelo y proporciona el contexto a las plantillas.
- **Controlador**: Es el código con la lógica que comunica el modelo con la vista.
- **Modelo**: Son los datos, que junto con la plantilla producen las vistas.
- **Vista**: Lo que el usuario visualiza. Parte de una plantilla, se junta con el modelo y se renderiza en el árbol DOM.





5. METODOLOGÍA

Para el desarrollo de este trabajo se ha seguido una metodología adaptada a las necesidades del proyecto, que coge elementos de otras metodologías existentes como Kanban o Scrum. No se ha podido elegir ninguna de estas metodologías en concreto debido a que el equipo de desarrollo solo tiene un miembro.

Para el proyecto se ha utilizado Trello como tablero donde colocar las tareas a realizar, y una serie de listas de acuerdo al estado de la tarea actual.

5.1. GitFlow + Trello

La gestión del proyecto desde el punto de vista del repositorio de versiones se ha realizado usando GitFlow modificado. Este tipo de organización de Git se basa en usar 2 ramas principales y 3 subgrupos de ramas. Las ramas principales son:

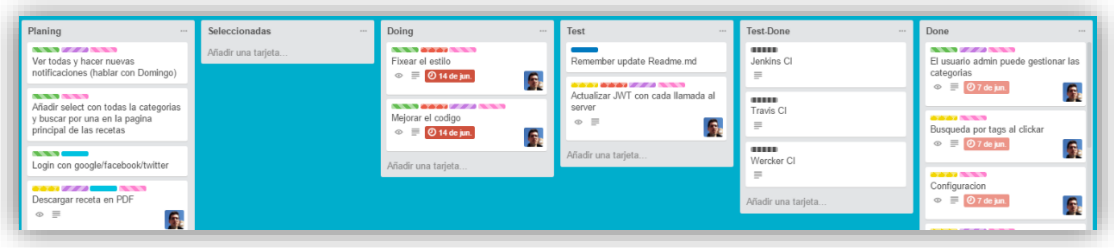
- **master:** en esta rama contiene el código que debe estar en producción
- **develop:** en esta rama es donde los miembros del equipo de desarrollo van integrando el código para la siguiente versión del proyecto.

Los 3 subgrupos de ramas, o ramas auxiliares son:

- **feature/***name*: en estas ramas se desarrollan nuevas características para la aplicación y, una vez terminadas y testeadas con el sistema de CI, se incorporan a la rama **develop**.
- **hotfix/***name*: en GitFlow, las ramas **hotfix** deben salir de **master** e incorporarse a **master** y **develop**. En el caso de este proyecto, al tratarse de un único desarrollador, los **hoftix** se realizan de **develop** a **develop** por comodidad.
- **release/***versión*: en estas ramas es donde se prepara el código y se corrigen los últimos fallos antes de pasar a producción.

Unir la gestión del proyecto con GitFlow con un tablero de Trello permite hacer un seguimiento de cada rama. En la siguiente imagen se puede observar esta organización:

Doing y Test	Test-Done	Done
Todas las tarjetas de estas listas deben ser ramas **feature** o **hotfix**	Todas las tarjetas de esta lista deben estar en la rama **develop**	Todas las tarjetas en esta lista deben estar en la rama **master**.
Para pasar a la siguiente columna, Test-Done, deben pasar los test del CI	Para pasar a la siguiente columna, deben ser aprobadas en la reunión del Scrum	



5.2. Trello + Scrum/Kanban

En este proyecto se han llevado a cabo técnicas de Scrum y Kanban. Se ha elegido la parte de Scrum relacionada con sprints y reuniones semanales, y la parte de Kanban en cuando a la gestión de tarjetas en Trello y reglas, como no iniciar nuevas tareas sin acabar las ya empezadas.

Las reuniones de Scrum se realizan con el tutor cada 2 semanas, por lo que los sprints de Scrum duran lo mismo. Gracias a Trello se puede ver si los plazos de las tareas se están cumpliendo correctamente con lo estimado.

5.2.1. Listas

Al principio del proyecto, se añaden todas las tareas en la lista Planing. En la reunión de Scrum, se debate y eligen las tarjetas que serán desarrolladas en el siguiente sprint. En esta reunión también se comprueba el resultado del sprint anterior, mirando las tarjetas de la lista Test-Done, y si todo es correcto se mueven a Done, incorporando las nuevas características a la rama **master**.

Como regla general, todas las nuevas tareas son añadidas en la lista Planing para ser propuestas en la siguiente reunión de Scrum, salvo que se traten de errores graves, los cuales deben ser arreglados lo antes posible y se añaden con Prioridad alta en la lista Seleccionadas y se deben realizar antes que cualquier otro desarrollo.

5.2.2. Tareas

Al principio del proyecto, las tareas se escriben de forma general, y dentro se añaden subtareas a realizar. También se le añade una prioridad, la parte de la aplicación que se ve afectada (API REST o Angular JS), el miembro responsable de la tarea y la fecha aproximada de vencimiento; todo ello usando los elementos de Trello explicados anteriormente.

A medida que el proyecto va avanzando, las tareas se vuelven cada vez más específicas, ya sea porque se trata de tareas nuevas añadidas posteriormente al planteamiento inicial como por errores y fallos durante el desarrollo.

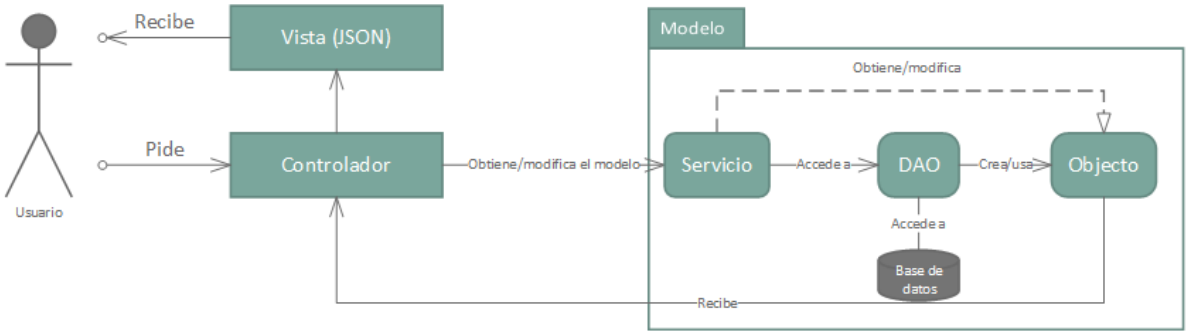
6. ARQUITECTURA

6.1. API REST

En la API REST se pueden distinguir varias capas:

- **Modelo**: Capa de acceso a datos. En ella se realiza la conexión con la base de datos y son devueltos en objetos al controlador. Dentro de esta capa se ha añadido otro patrón llamado DAO (Data Access Object) la cual tiene los métodos necesarios para gestionar un modelo, permitiendo aislar el acceso a los datos del resto de la aplicación.
- **Servicio**: contiene todos los métodos DAO y añade métodos extra combinando varios modelos, como por ejemplo al añadir una etiqueta a una receta.
- **Controlador**: Aquí encontramos la lógica de la aplicación. Esta capa se encarga de recuperar la información introducida por el usuario y realizar las acciones pertinentes llamando al modelo. Después envía la información a la capa de la vista.

El esquema final sería algo así:



6.2. Cliente Web

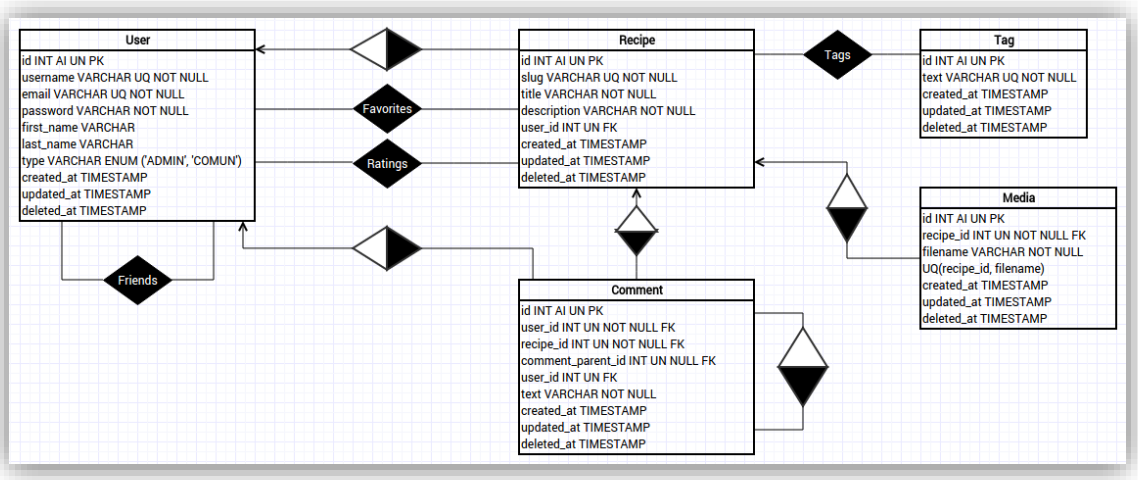
Para el cliente Web en AngularJS se ha escogido el patrón MVC (modelo-vista-controlador) como patrón principal de desarrollo. Este patrón separa una aplicación en 3 capas:

- **Modelo**: Para AngularJS, un service equivaldría a un modelo estándar. En este service se definen los métodos que usaran los controladores para obtener y/o modificar los datos. También AngularJS posee un módulo para poder generar modelos en JavaScript llamado ngResource que automatiza todo pero debido a que algunas rutas son diferentes al estándar no se usará.
- **Vista**: Aquí se muestra la información al usuario usando las plantillas HTML.
- **Controlador**: Los controladores se encargan de recoger los datos de los modelos y enviárselos a las vistas para que estas puedan renderizarlos. Esto se realiza a partir de la variable $scope que nos proporciona Angular.

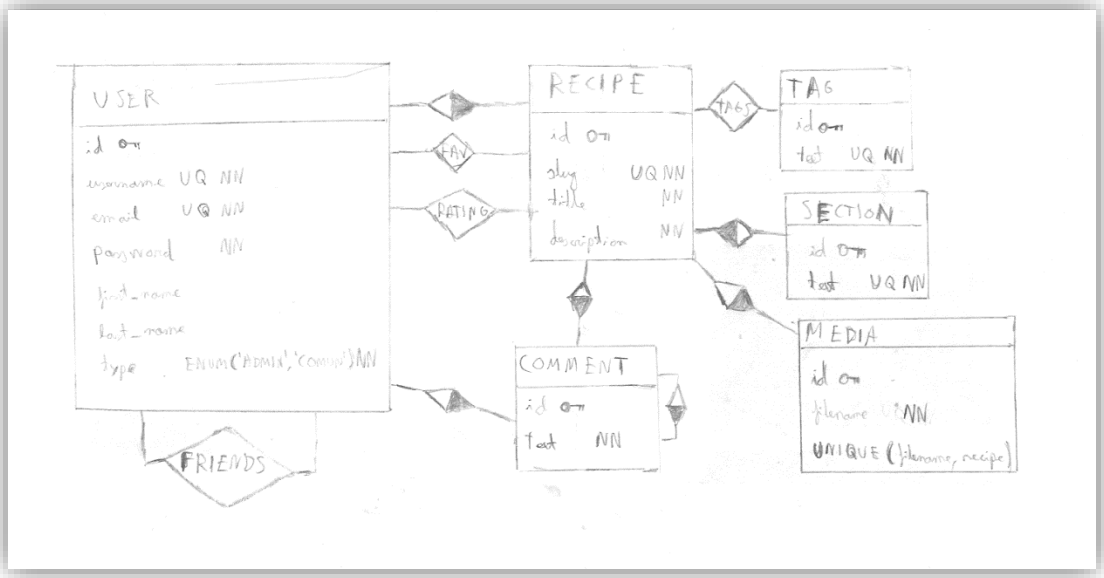
6.3. Esquema de la Base de Datos

La base de datos ha pasado por varias fases, debido a cambios estructurales y refactorizaciones varias.

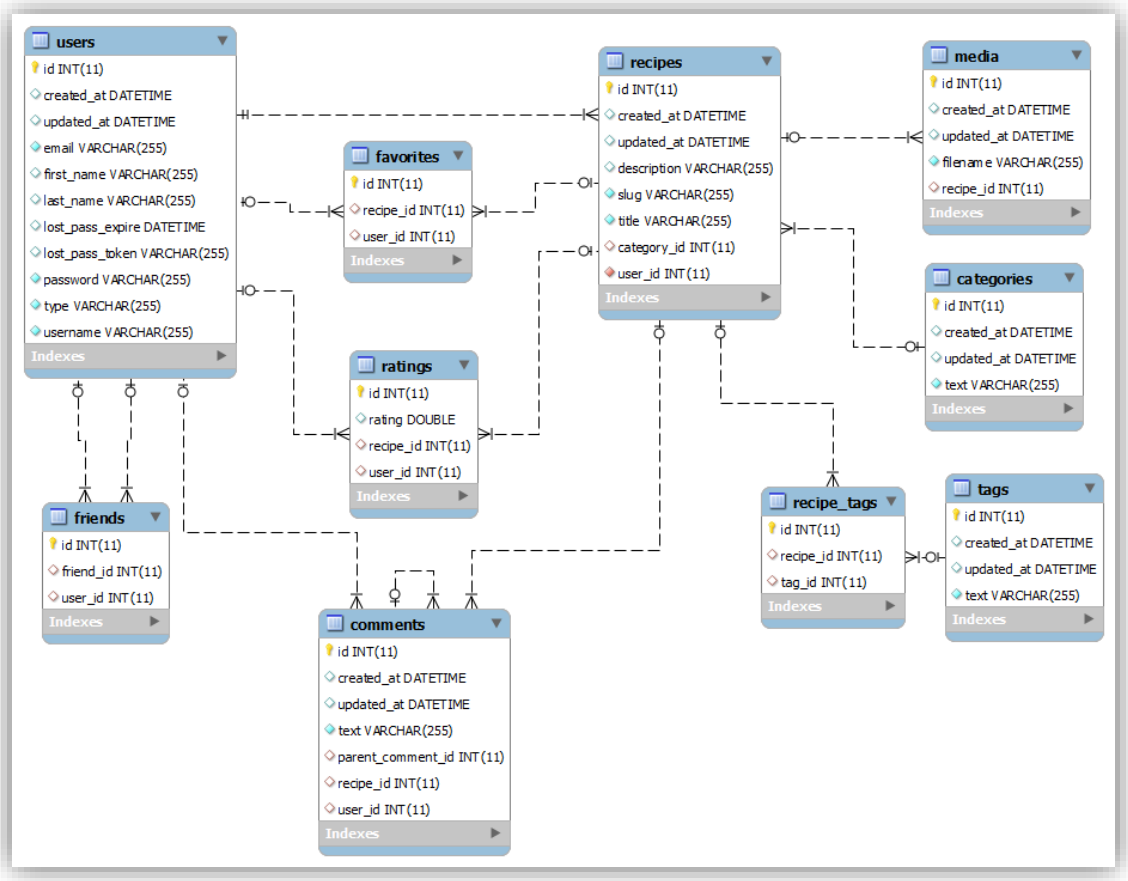
6.3.1. Versión inicial



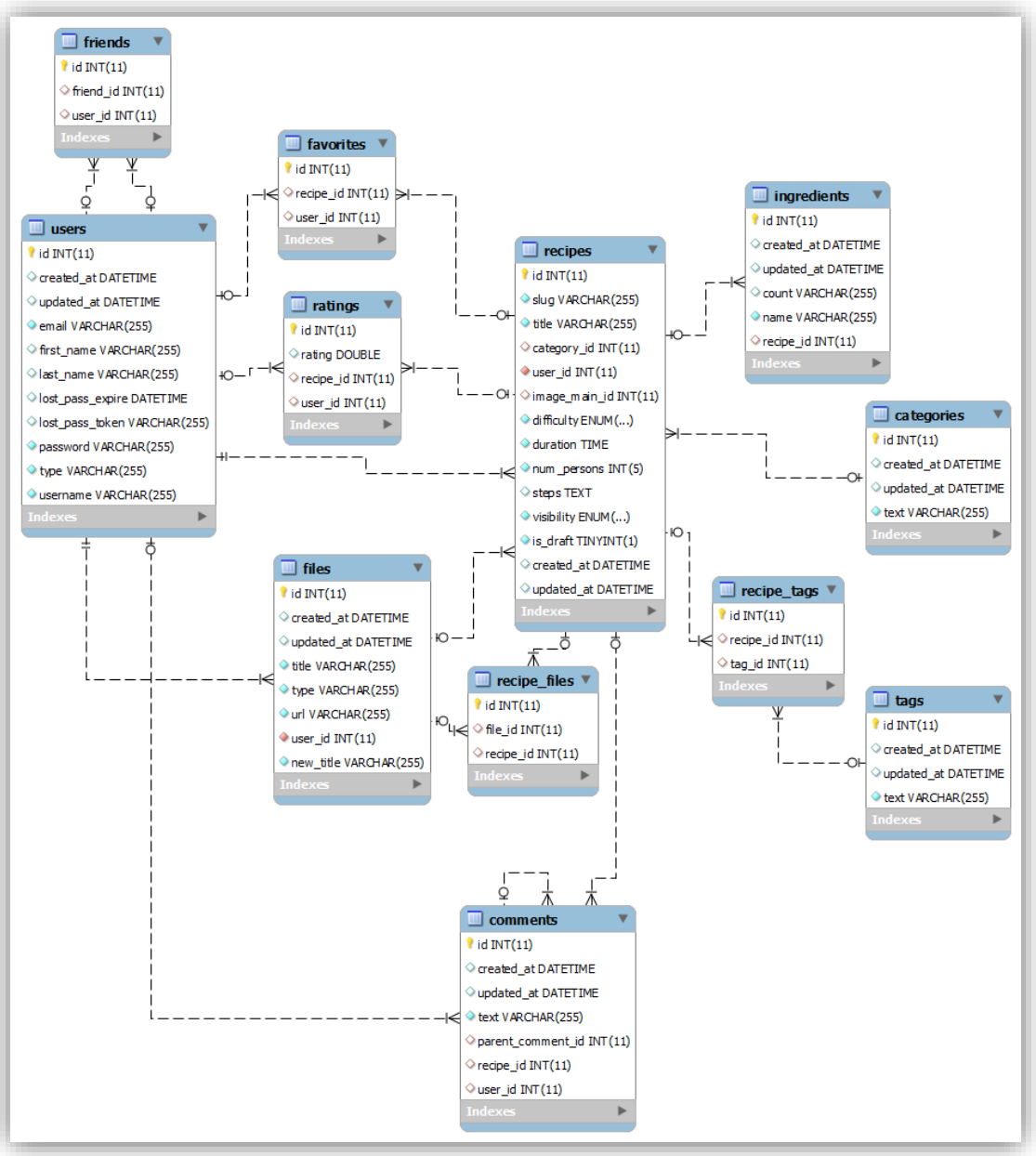
6.3.2. Versión 2: Añadir tabla ‘sections’



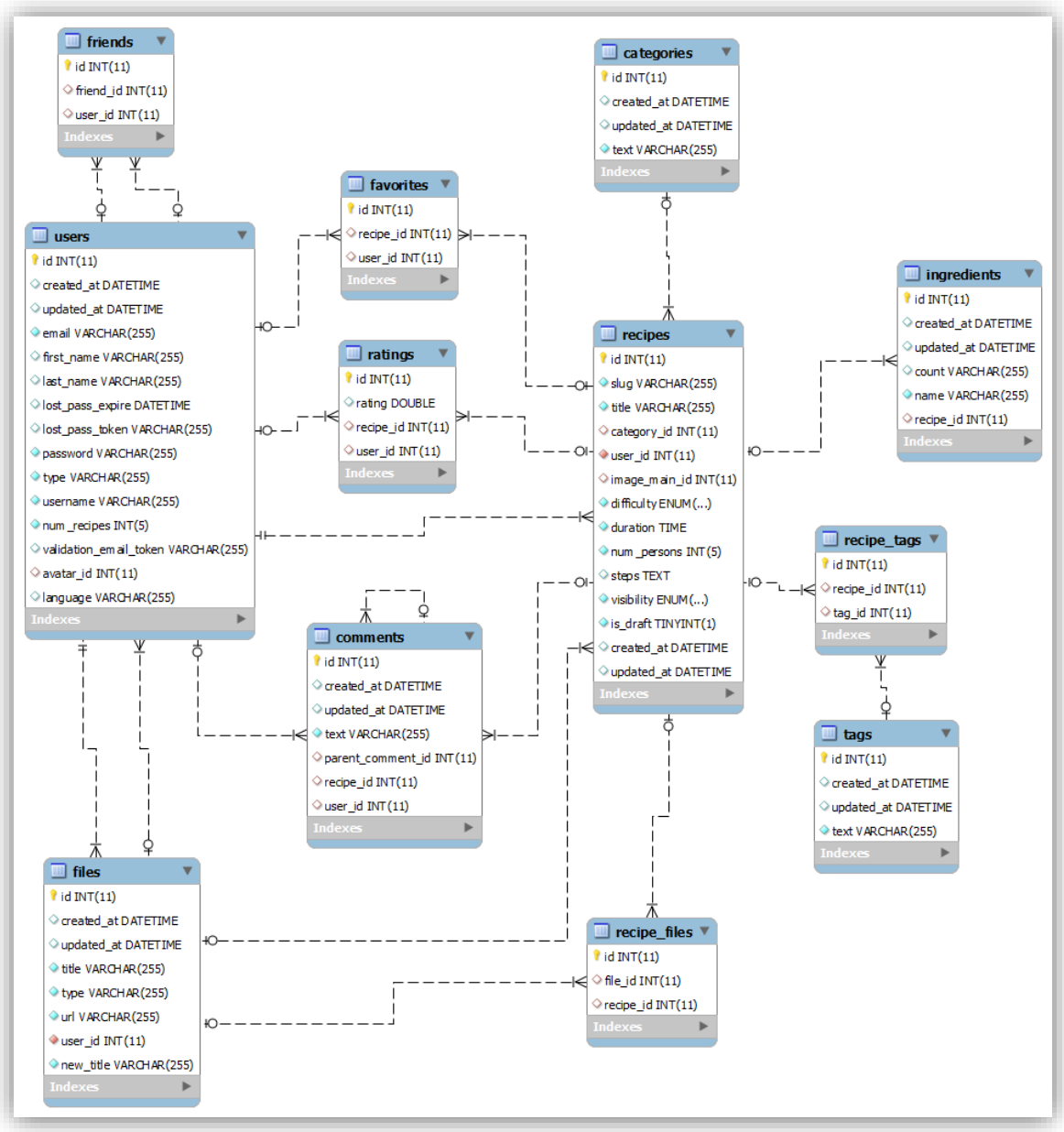
6.3.3. Versión 3: Refactorización tabla ‘sections’ a ‘categories’



6.3.4. Versión 4: Refactorización de las tablas relacionadas con archivos

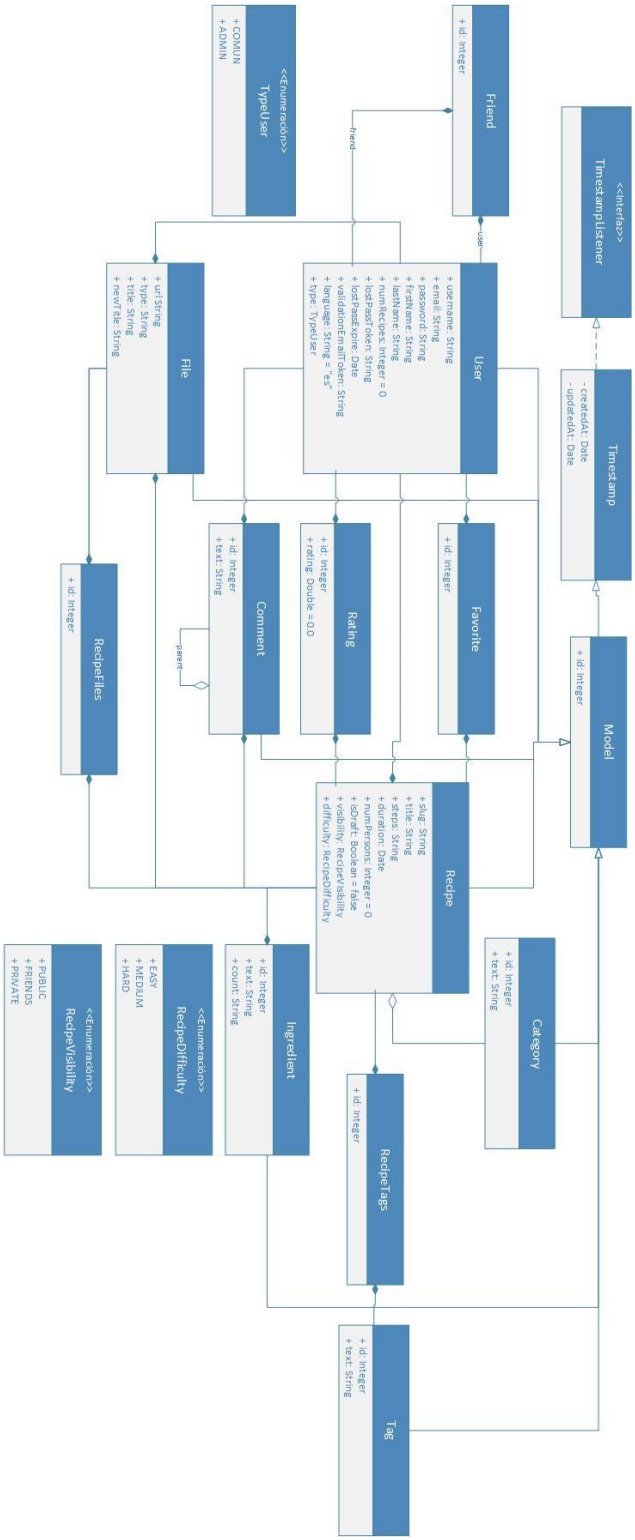


6.3.5. Versión 5: Añadidos nuevos campos extra a los usuarios



6.4. Diagrama de clases

A continuación se puede apreciar el diagrama de clases usado para la API REST. Gracias a JPA el modelo de datos se genera automáticamente a partir de los objetos Java.



7. VISTAS Y FUNCIONALIDADES

Anteriormente, en el apartado Objetivos, se han mencionado las funcionalidades principales que han sido propuestas. Tales funcionalidades son:

- Permitir el registro e identificación de usuarios. Los datos sensibles se almacenan en la base de datos, y el sistema provee un JWT para autentificar el acceso de forma segura a partes privadas.
- Buscar recetas por etiquetas y texto. Aparte, el sistema permite ordenar y paginar el resultado
- Ver detalladamente recetas. Dentro de ella se podrá añadir a favoritos, puntuar y comentar recetas y responder a otros comentarios.
- Gestionar recetas. El usuario tendrá completo acceso a sus recetas, permitiendo en cualquier momento actualizar los datos de cualquier receta.
- La web estará adaptada tanto para PC como para móviles. Esto permite que el usuario pueda usar la aplicación en cualquier dispositivo, independientemente del hardware.
- También el sistema ha sido completamente traducido en dos idiomas: español e inglés.

Al tratarse de un proyecto basado en una API REST con un front-end separado, la API esta independizada de la parte de presentación al usuario, lo que permite ser escalable con mayor facilidad.

7.1. API REST

Todas las funcionalidades de la API REST están descritas aquí: https://recetarium.herokuapp.com. A continuación se van a describir las más importantes.

7.1.1. Listado de recetas

Las recetas son la base de este proyecto, por lo que el listado de ellas es importante a la hora de mostrar cada una con los detalles necesarios. Esta petición de la API se hace con la siguiente configuración:

Definición			

`GET`	`/recipes`	`page: int`
		`size: int`
		`search: string`
		`order: string`
		`tags: array de strings`
			`page: 1`
			`size: 10`
			`search: “”`
			`order: “-created_at”`
			`tags: []`

- El campo *order* solo acepta como válidos los campos básicos de una receta
- El buscador busca tanto en el titulo como en los pasos de la receta
- El campo *tags* permite añadir múltiples valores, lo que acota la búsqueda a las recetas que tengan ambas etiquetas: tags=2&tags=3

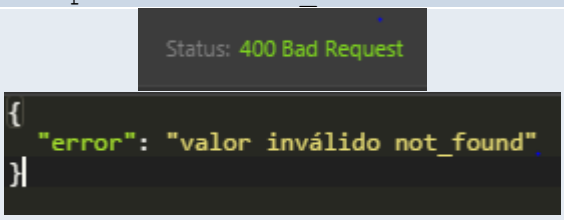
Ejemplos

/recipes

/recipes?page=2&size=3&search=recipe&order=id

/recipes?tags=9

/recipes?order=not_found



7.1.2.

7.1.3. Login de un usuario

La funcionalidad de login es una de las bases de las aplicaciones web en general. Para abordar esta funcionalidad, se ha propuesto es siguiente endpoint:

Definición			
POST	/auth/login	email: string password: string setExpiration: bool	email: null password: null setExpiration: false

- Tanto el *email* como la *password* son campos necesarios
- El campo *setExpiration* solo afecta al front-end, el token tiene siempre 30 minutos de duración
- El *endpoint* devuelve 3 campos: el JWT, la clave de pusher para las notificaciones del usuario, y el lenguaje del usuario que se ha logueado.

Ejemplos

/auth/login (sin campos)

```
{
    "email":"",
    "password":"",
    "setExpriation": false
}
```

Status: 400 Bad Request

```
{
  "password": [
    "esta campo es obligatorio"
  ],
  "email": [
    "esta campo es obligatorio"
  ]
}
```

/auth/login (usuario no existe, usuario no activado, o contraseña incorrecta)

```
{
    "email":"no-existe@gmail.com",
    "password":"password",
    "setExpriation": false
}
```

Status: 401 Unauthorized

/auth/login (correcto)

```
{
    "email":"jvortsromero@gmail.com",
    "password":"password",
    "setExpriation": false
}
```

```
{
  "auth_token": "eyJhbGciOiJIUzI1NiJ9
      .eyJqdGkiOiJxSmJOdXFrVER5a1hyOEZiVFFGMG
      bmFtZVwiOlwiam9zcm9tXCIsXCJlbWFpbFwiOlw
      3J0cyBSb21lcm9cIixcInR5cGVcIjpcIkFETUlO
      RlZF9hdFwiOjE0NjQ3MjE2MjAwMDAsXCJsYW5nd
      lfSIsImV4cCI6MTQ2NjcyNTAzNX0.QA66xBcMct
  "pusher_key": "c0d58893e757f42fedf3",
  "language": "es"
}
```

7.2. Front-end

La aplicación de front-end es la encargada de hacer de intermedio entre las funcionalidades de la API REST y el usuario. El usuario que puede interactuar con la aplicación puede ser anónimo, autentificado o administrador.

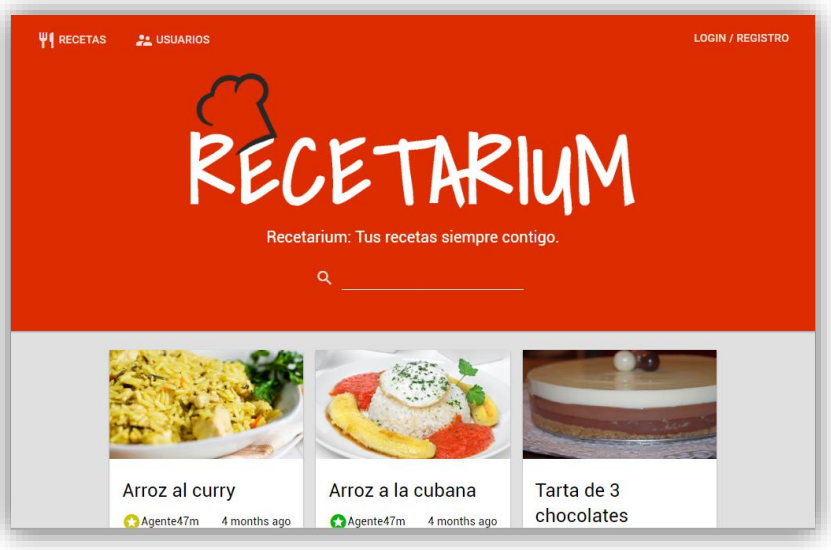
Todas las vistas se han realizado de forma que sean responsive, es decir, son adaptable a cualquier tamaño de pantalla de cualquier dispositivo. A continuación se van a describir las principales vistas para cada usuario.

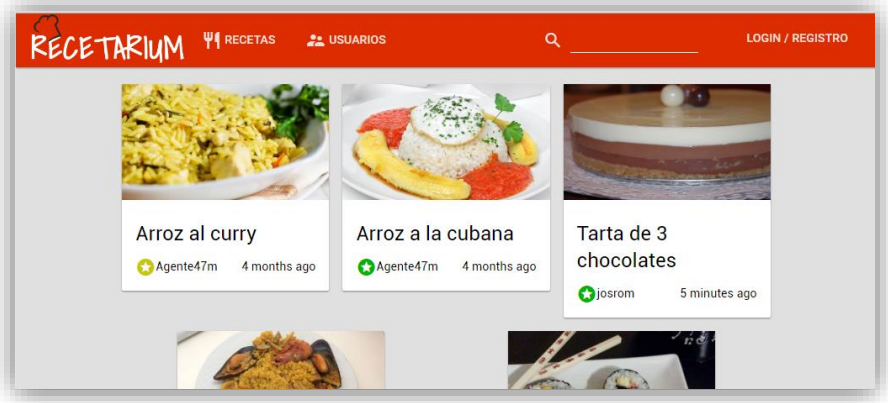
7.2.1. Como usuario anónimo

Un usuario anónimo es todo aquel que no está logueado. Por ello, la aplicación solo se muestra en modo lectura, sin poder cambiar ningún dato importante del sistema.

7.2.1.1. Home

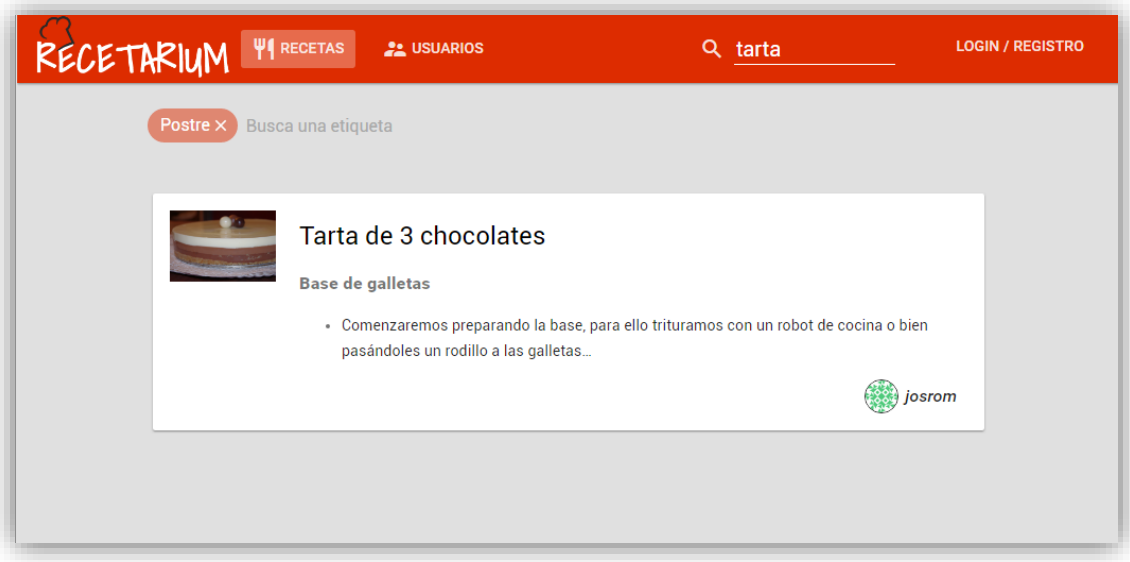
La pantalla principal de la aplicación muestra un listado de las recetas recientes, junto al buscador. También aparece en la barra superior el link para ir a la página de login. Esta pantalla tiene la peculiaridad de que al hacer scroll vertical el menú superior se cambia por otro más pequeño para que sea menos molesto para el usuario, pero manteniendo todas las funcionalidades.





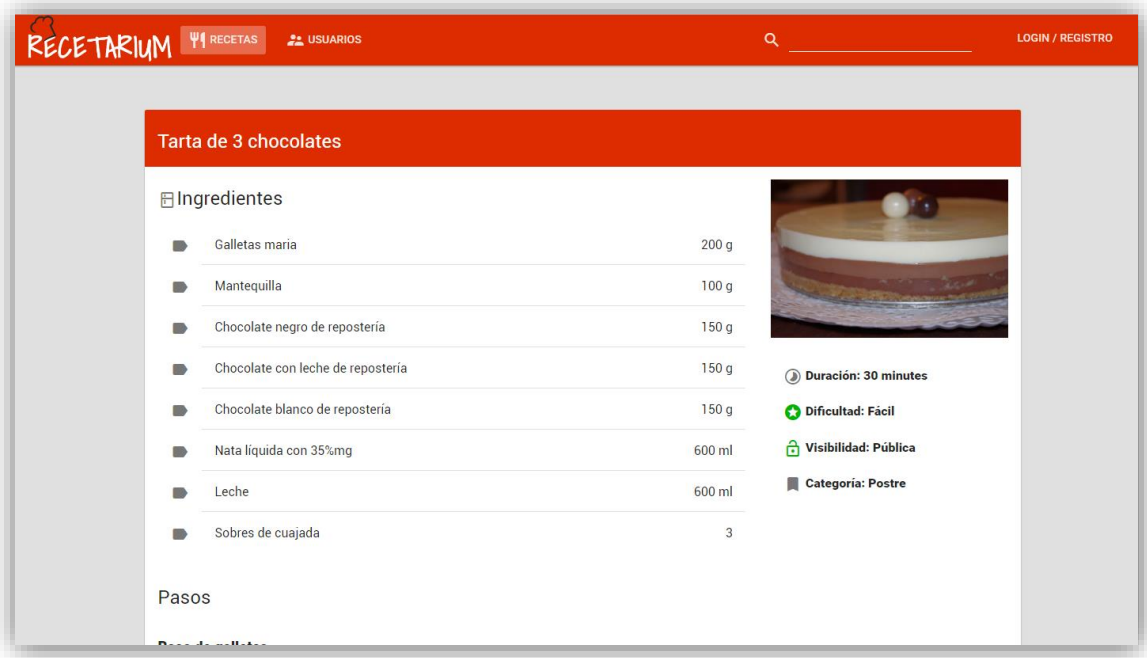
7.2.1.2. Recetas

Al hacer click en Recetas o en el buscador, la aplicación nos dirige a la vista con todas las recetas. Las recetas se muestran usando un infinite scroll, el cual va cargando las recetas de 10 en 10. También se puede buscar por etiquetas también.



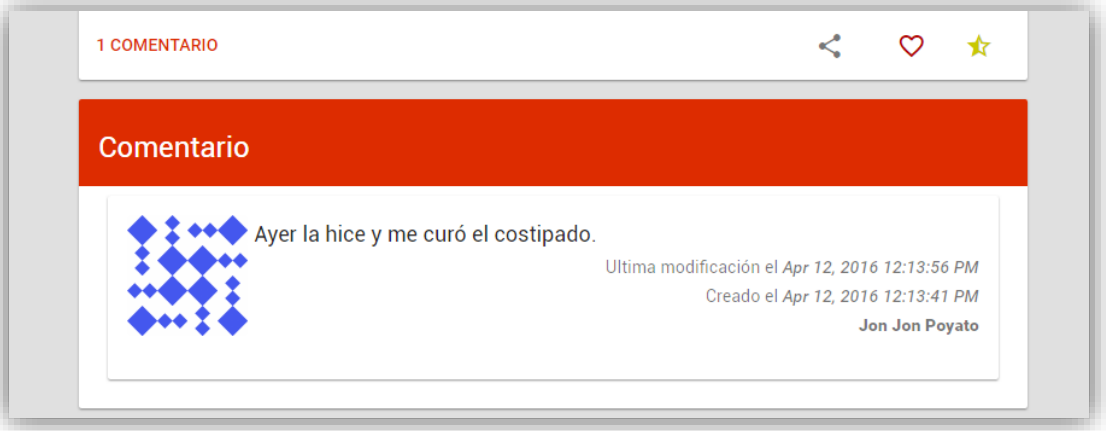
7.2.1.3. Receta

Dentro de una receta se pueden observar todos los detalles de la misma. Aquí se encuentran todos los ingredientes, la duración, la facilidad, etc.

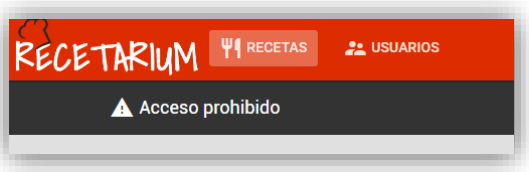


También se encuentra una galería de imágenes donde el autor puede añadir cualquier imagen para detallar la receta.

Por último, se encuentran los comentarios, los cuales son visibles al hacer click en el texto *N comentarios.*

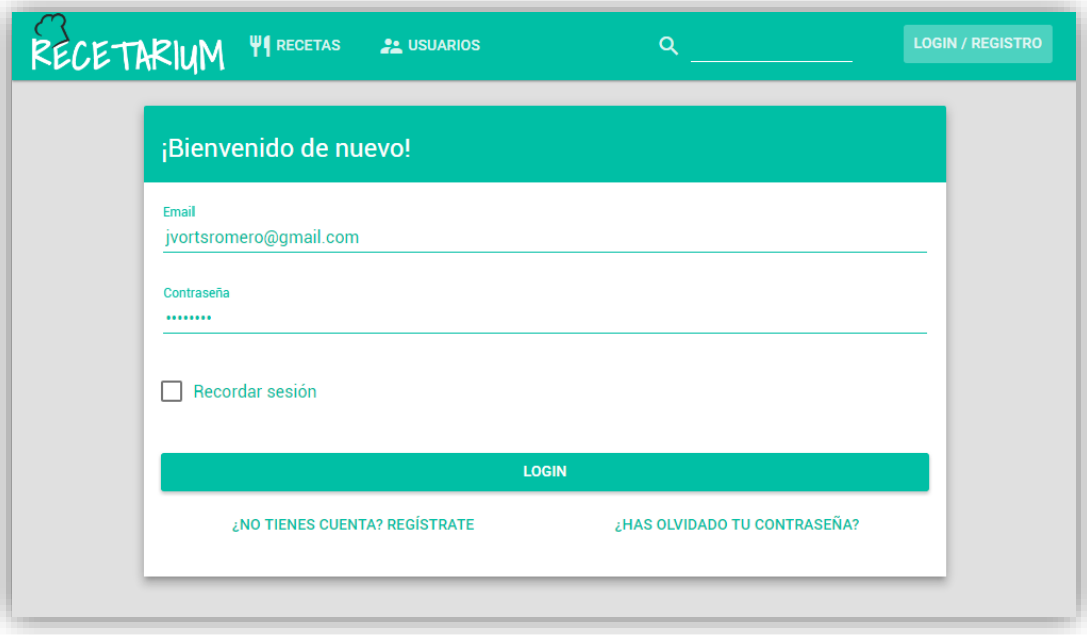


Si la receta no existe o no se puede acceder a ella porque no es publica, el sistema avisa al usuario con un mensaje de error:

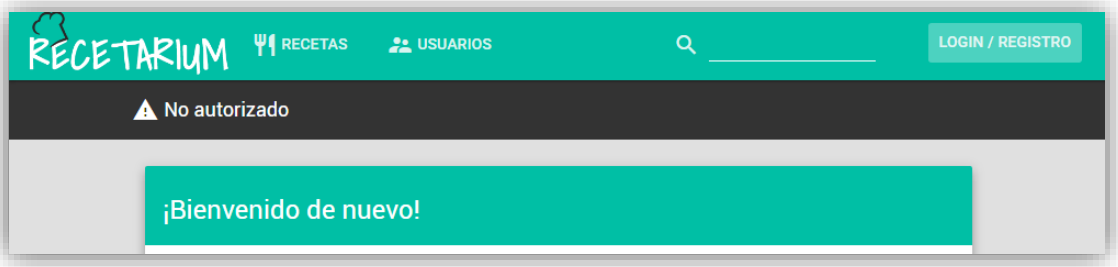


7.2.1.4. Login/registro

La última zona que pueden visitar los usuarios anónimos son las pantallas de login, registro o recuperar contraseña. Para separar las distintas zonas se han usado paletas de colores diferentes.



Si ocurriese algún problema en algún formulario, un mensaje de error aparecerá justo debajo de la barra superior:

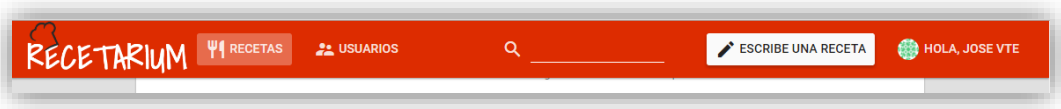


7.2.2. Como usuario autentificado

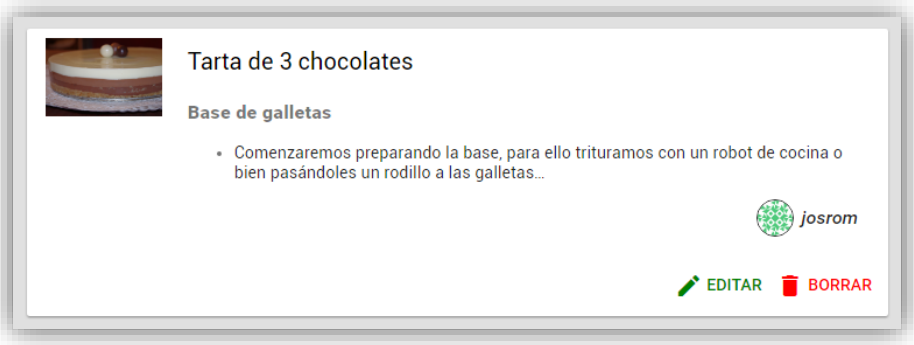
Un usuario anónimo se convierte en autentificado cuando el formulario de login es rellenado y enviado correctamente, y el JWT es guardado en el navegador. Este usuario tiene varias funcionalidades extras, como la creación, edición y borrado de recetas, o la capacidad de comentar en las recetas.

7.2.2.1. Gestión de las recetas del usuario

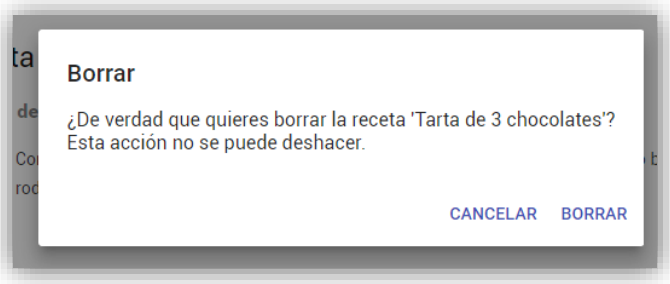
Para acceder a la creación de recetas se puede pulsar tanto el botón de la barra superior como el botón inferior en la página de recetas.



Para acceder a la edición de recetas se puede mediante el botón Editar que aparece en la lista de recetas. Desde esa misma lista también aparece el botón de Borrar.

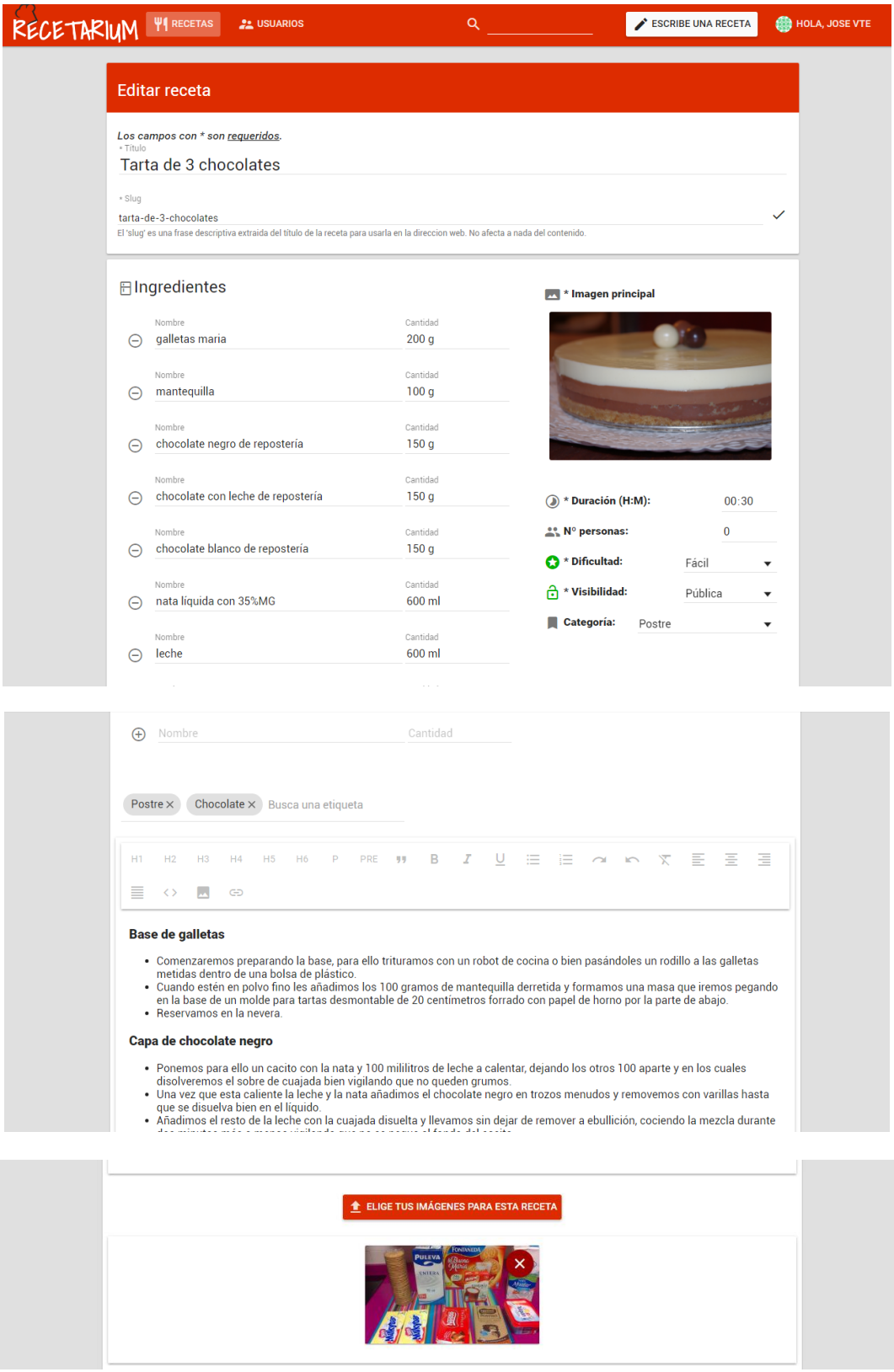


Al borrar una receta se necesita confirmación, para evitar algún missclick:

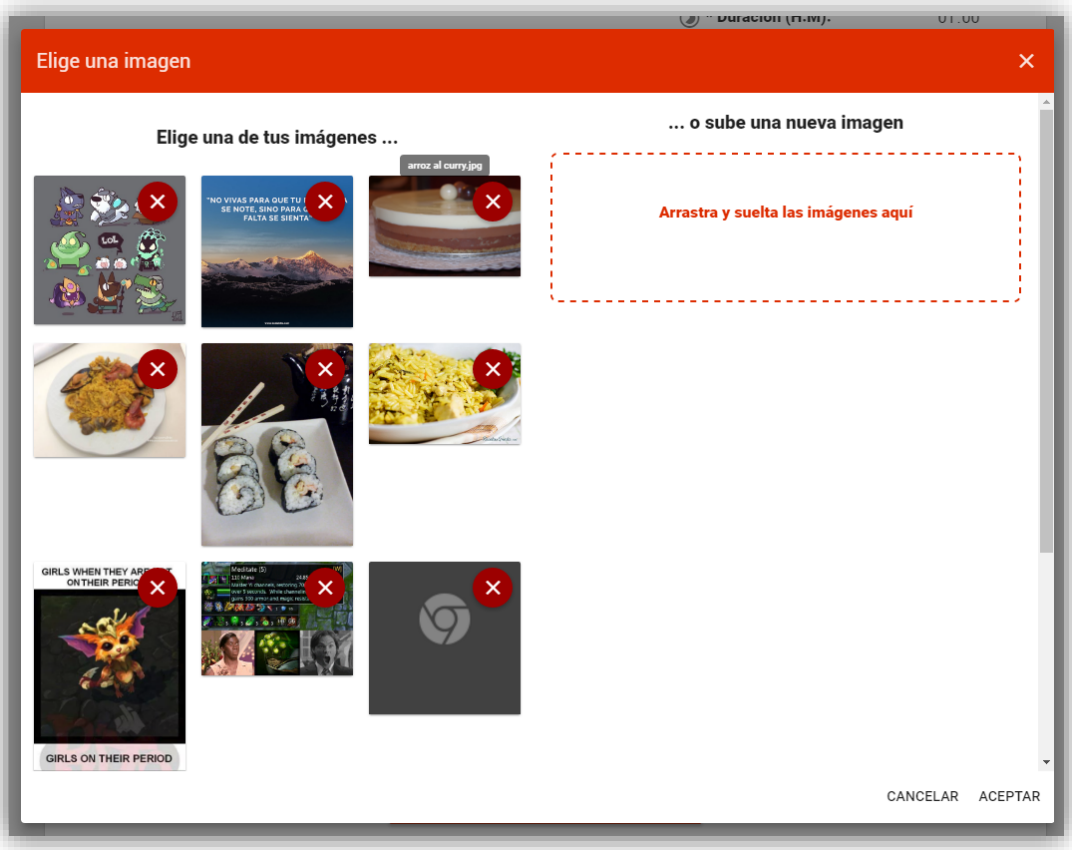


Las paginas para crear y editar una receta son iguales, salvo por los botones que están debajo de la página, ya que al estar creando una receta nueva puedes guardar el progreso actual o publicarla cuando todos los datos estén correctos.

Aquí se puede apreciar toda la página de edición completa:

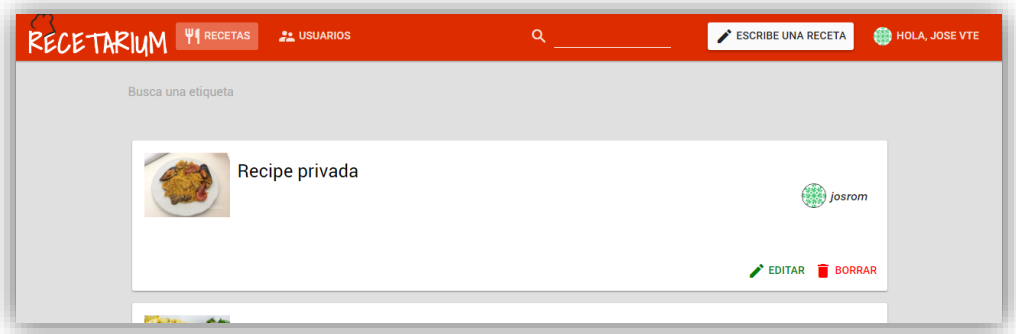


La aplicación tiene un sistema basado en una galería de imágenes para poder añadirlas a las recetas:



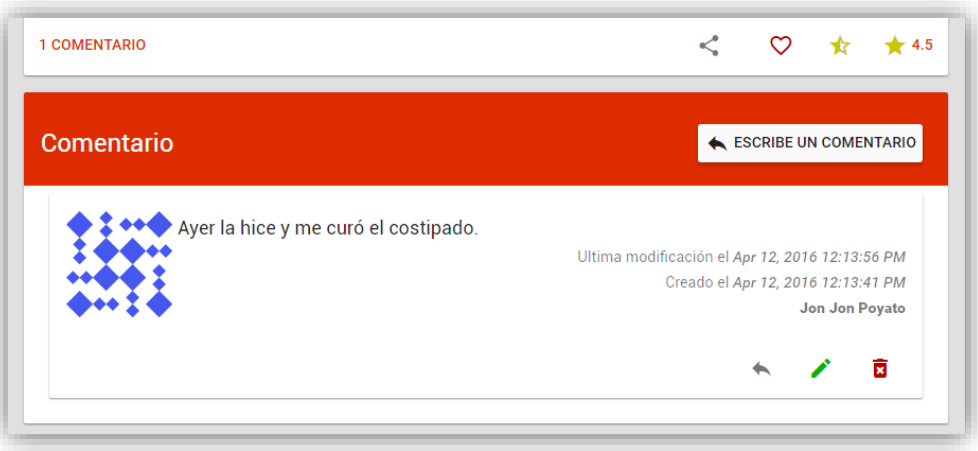
7.2.2.2. Recetas privados y de amigos

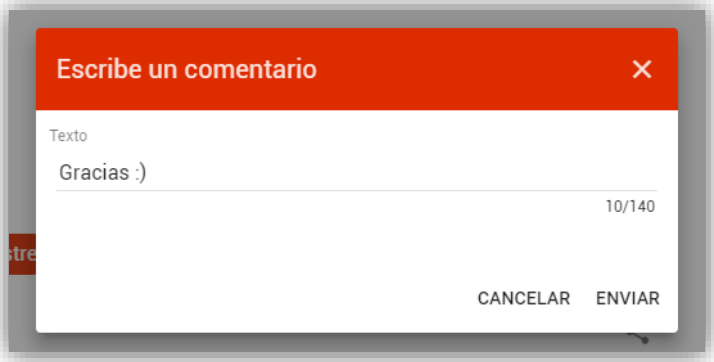
Ahora en el listado de recetas aparecen tanto las recetas de los amigos que tienes como las recetas privadas tuyas.

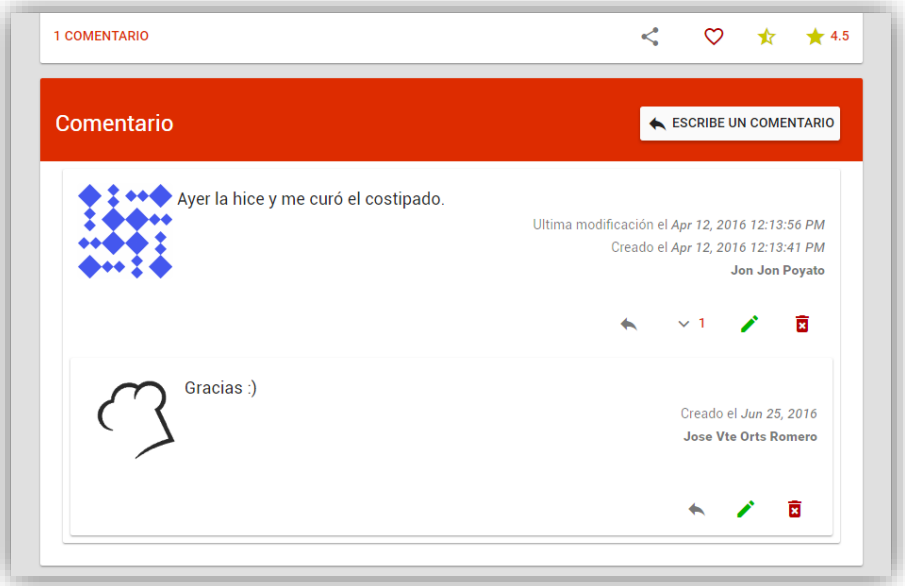


7.2.2.3. Comentarios, favoritos y puntuación

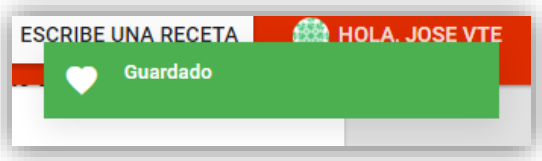
Dentro de una receta se pueden añadir comentarios y responder a ellos.

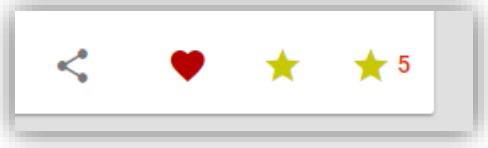






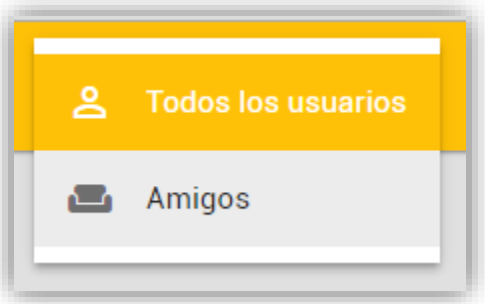
También se puede marcar como favoritos y puntuar una receta.

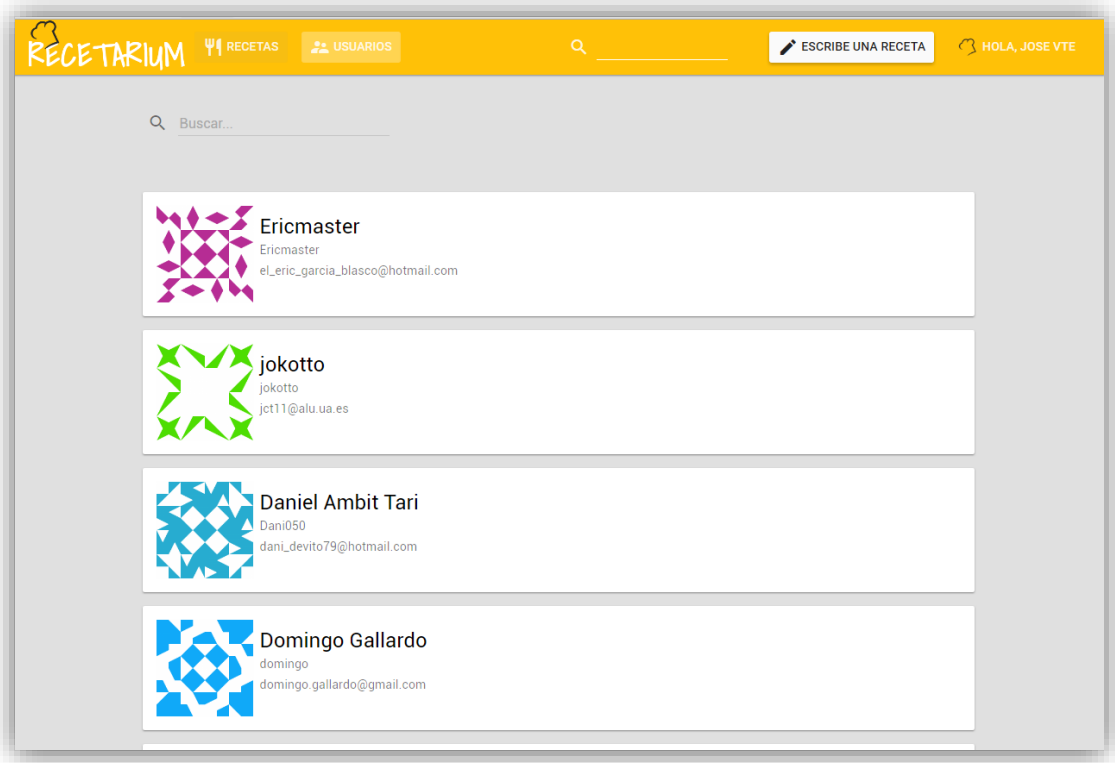


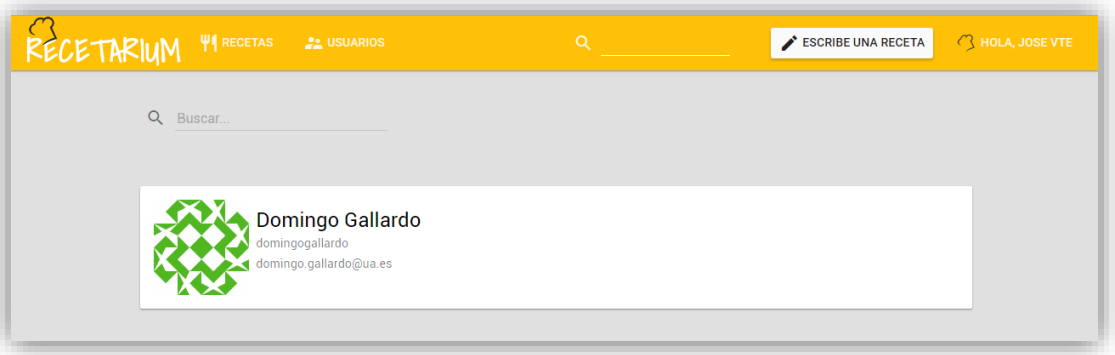


7.2.2.4. Usuarios y amigos

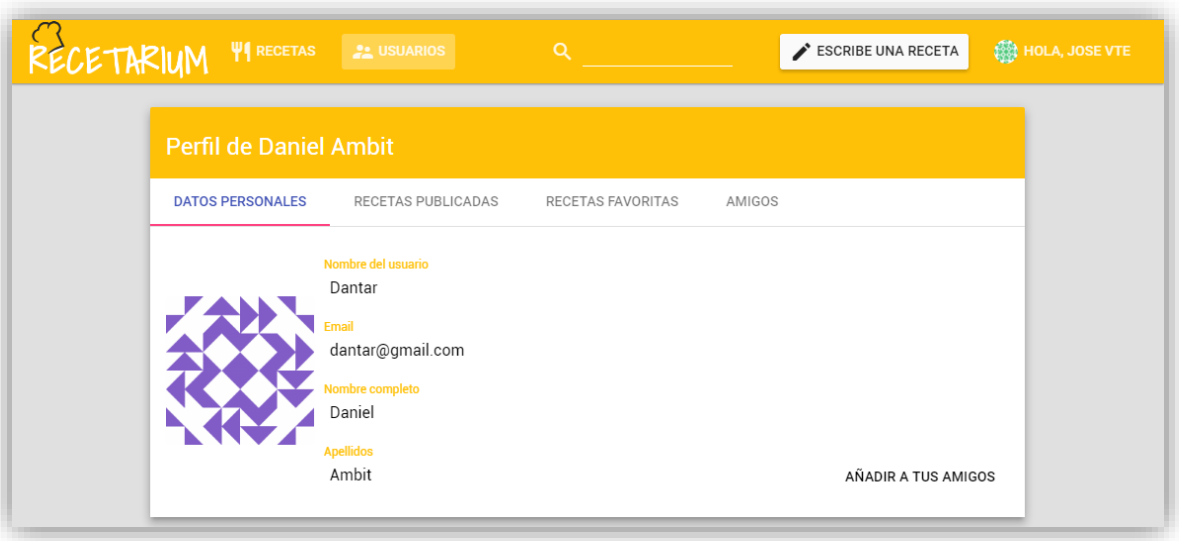
La sección de usuarios y amigos aparece un listado, al igual que en recetas, con todos los usuarios que hay en la aplicación.





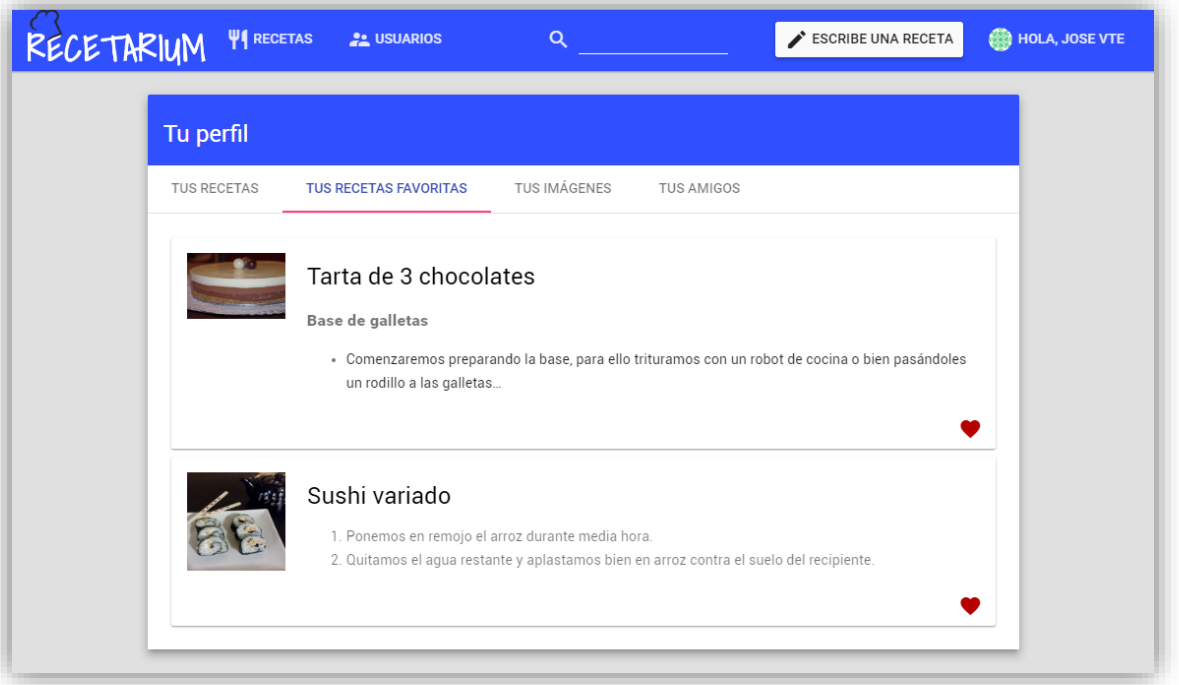


Al hacer click en alguno, se muestra la pantalla del perfil de ese usuario, junto con las recetas, los favoritos y los amigos. El usuario también puede añadirlo a sus amigos de forma fácil e intuitiva con el botón *Añadir a tus amigos*.

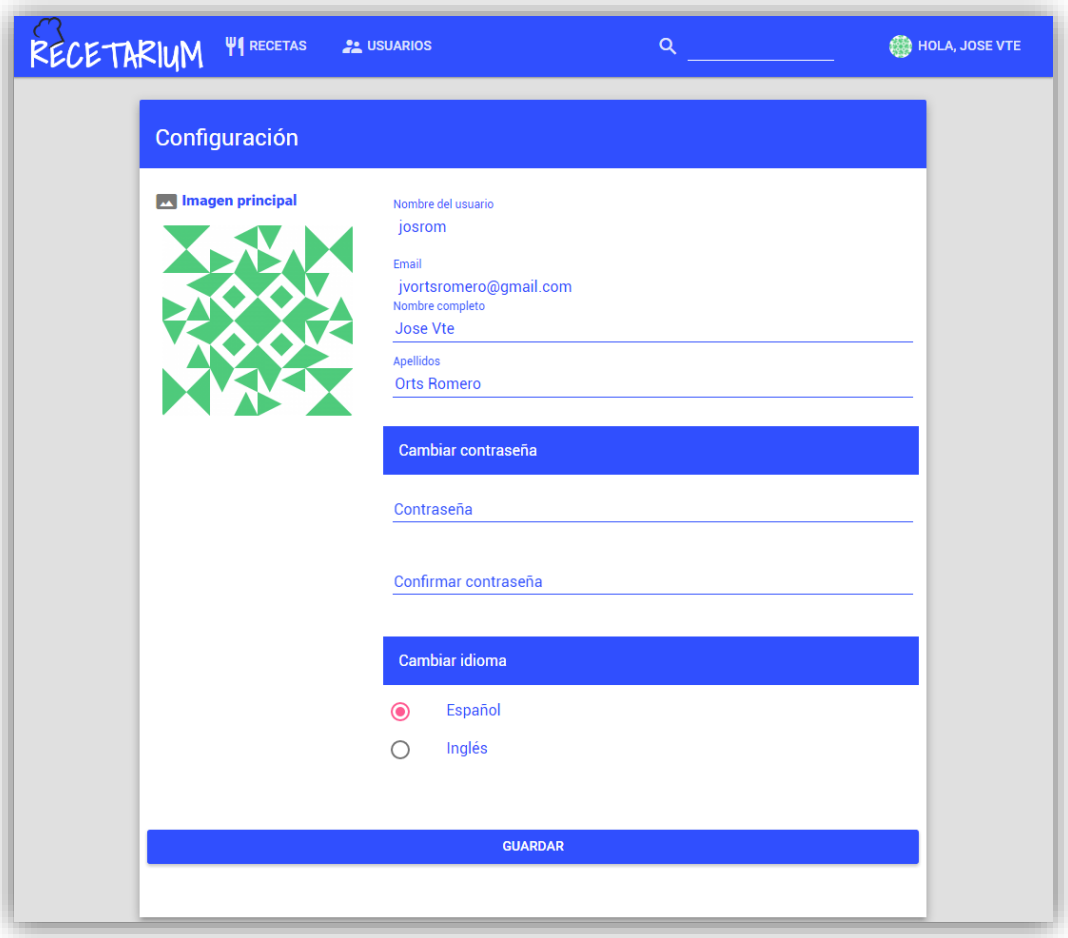


7.2.2.5. Perfil y configuración

En la barra superior aparece el nombre del usuario, junto con un menú desplegable. En él se puede acceder al perfil, donde se encuentran las recetas, las imágenes, las recetas favoritas y los amigos del usuario.



En configuración, el usuario puede cambiar sus datos, la foto de perfil y el idioma de la aplicación.

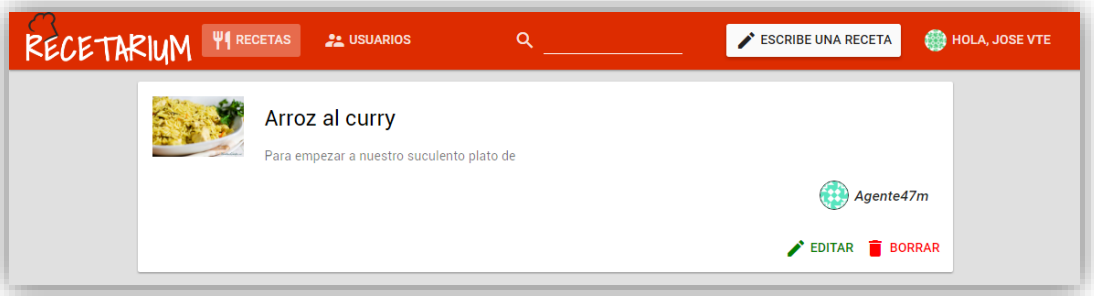


7.2.3. Como usuario administrador

Un usuario administrador puede hacer cualquier funcionalidad descrita antes y, además, gestionar otro tipo de recursos del sistema. Para crear un usuario administrador, por ahora se ha de usar el acceso de la API y crear un usuario o editar uno ya existente cambiándole el tipo de COMUN a ADMIN.

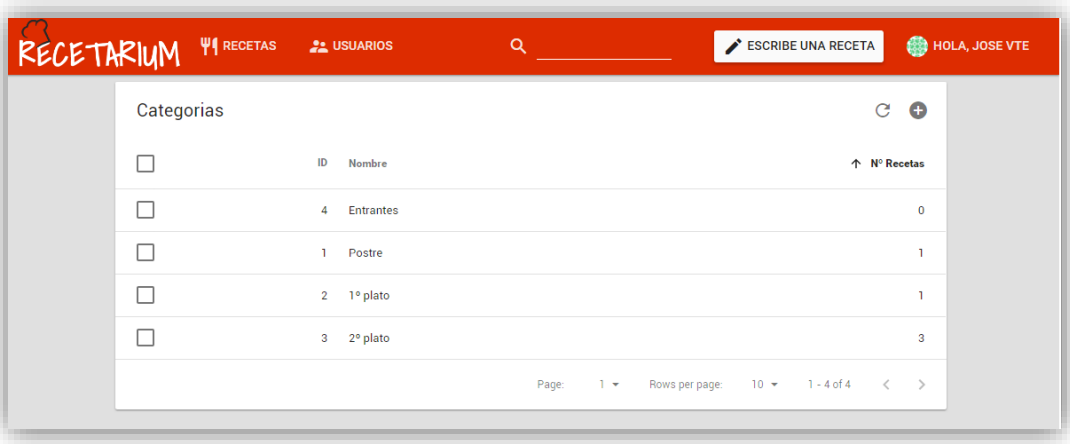
7.2.3.1. Gestión de recetas de otros usuarios

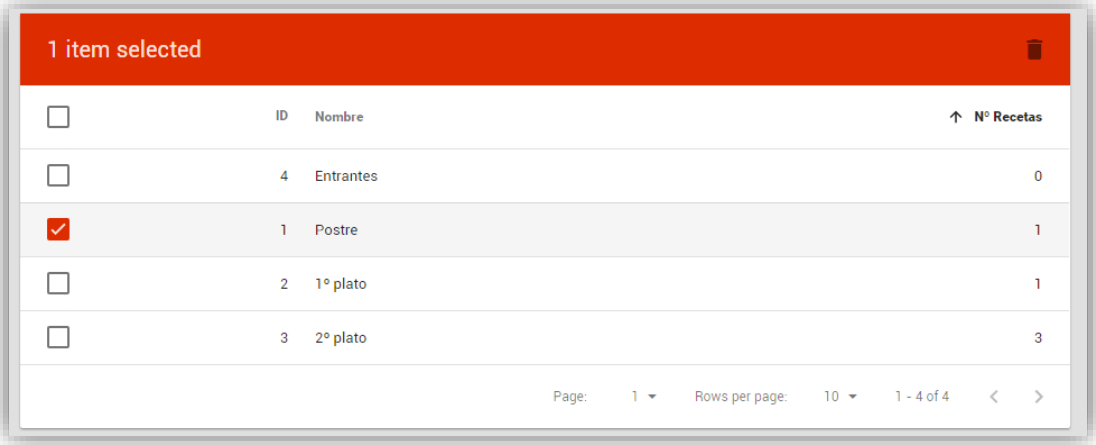
Los usuarios administradores pueden gestionar recetas de otros usuarios, mediante los mismos botones que el usuario normal.

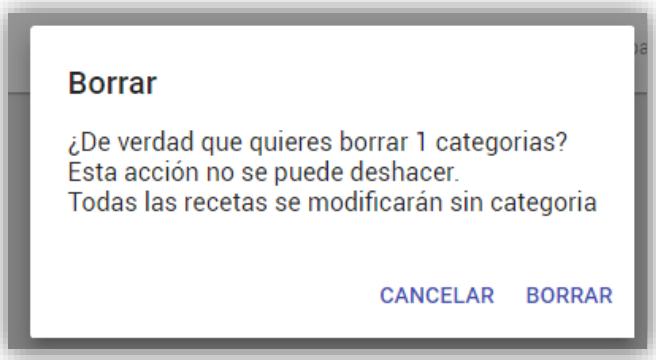


7.2.3.2. Gestión de categorías

La pantalla de gestión de categorías permite una visualización de todas para un análisis rápido, ya que aparece el número de recetas total con esa categoría. En esta pantalla se ha usado un datatable con paginación en vez de un infinite scroll porque los datos de gestión deberían ser lo más editables posibles.





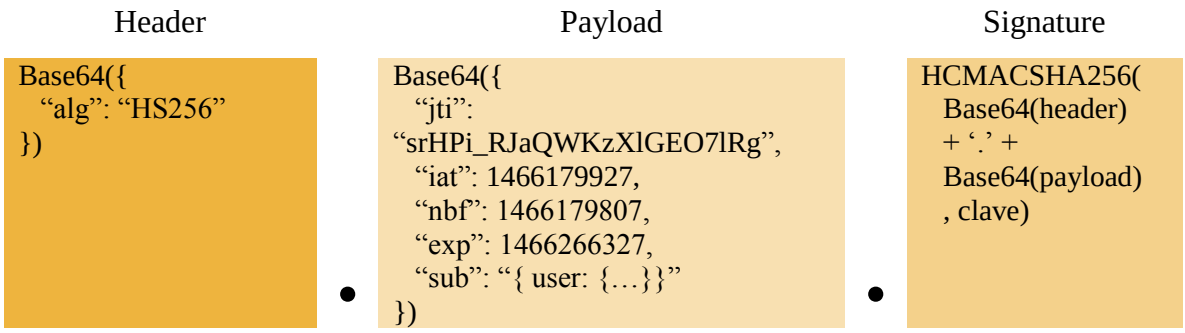


8. SEGURIDAD

La seguridad en un proyecto Web es muy importante, tanto separar la parte privada de la pública como evitar el acceso a la parte de administración a usuarios no autorizados.

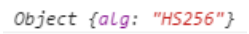
8.1. JWT

Para la comunicación entre el front-end y la API REST se ha usado en JWT (JSON Web Token). La estructura de un JWT se podría dividir en 3 partes *header*, *payload* y *signature*, las cuales se cifran en base64 y unen mediante un punto.



8.1.1. Header

En la parte de la autenticación se define el algoritmo que se usa para comprobar el Token:



8.1.2. Payload

En la parte *payload* se definen los campos que identifican al usuario del Token, el cuándo se ha creado, la duración, etc. Todos los campos son creados en la API REST, en el front-end solo se lee el Token y se descifran los datos llegados.

Token descifrado usando base64 de JS

Como se puede observar, los campos pueden contener cualquier dato. Existe un estándar definido en Internet para el uso de los JWT:

- **jti** (JWT ID): identificador único.
- **iat** (Issued at): fecha en la que se emitió el JWT.
- **nbf** (Not before): fecha a partir de la cual el token empieza a ser admitido.
- **sub** (subject): identifica el tema del token.
- **exp** (expiration time): fecha de expiración del token, a partir de la cual ya no es válido.

8.1.3. Signature

Usando el algoritmo definido en la parte de la autenticación y la clave del servidor, se cifra la unión del header más payload y se concatena al string previo en base64, quedando así la estructura final del JWT.

Esta última parte es la que confiere la seguridad al JWT, ya que cuando el Token le llega al servidor, descifra el mensaje y comprueba que los datos sean válidos y que no haya expirado el Token.

8.2. Permisos

La aplicación consta de un sistema de permisos para acceder a ciertas zonas, y así poder separarlo en parte publica, parte privada y zona de administración. Un usuario tiene definido y guardado en la base de datos el tipo que es: COMUN o ADMIN.

- Como usuario anónimo o no logueado no se puede editar nada, solo se pueden ver las recetas públicas. Es el modo más simple de la aplicación.
- Como usuario normal puedes acceder a nuevas zonas, como tu perfil, recetas privadas o compartidas con amigos, etc., aparte de las zonas públicas.
- Como usuario administrador puedes acceder a las mismas zonas que un usuario normal, y también a la zona de administración. En esta zona por ahora solo están las categorías. Otra peculiaridad de los administradores es que pueden editar y borrar las recetas y comentarios de otros usuarios, por si hubiese algún tipo de abuso o falta de respeto.

8.3. Otras técnicas

- La contraseña del usuario se ha guardado usando una librería de Java para BCrypt más una sal. Dicha librería permite luego comprobar la contraseña fácilmente.
- Si alguna respuesta de la API REST devuelve un código HTTP 401 o 403, AngularJS redirige automáticamente a una página donde avisa al usuario de que no está autorizado o le faltan permisos respectivamente.
- Si el usuario ha marcado la casilla de “Recordar sesión”, el sistema automáticamente cada 30 minutos hace una comprobación contra la API REST para renovar el JWT.
- Tanto el Token como el subject del Token descifrado se guardan en una variable del Local Storage y en el $rootScope de AngularJS cuando el usuario se autentifica.
- AngularJS posee un módulo, **sanitize,** para comprobar la entrada y salida de datos de las variables.

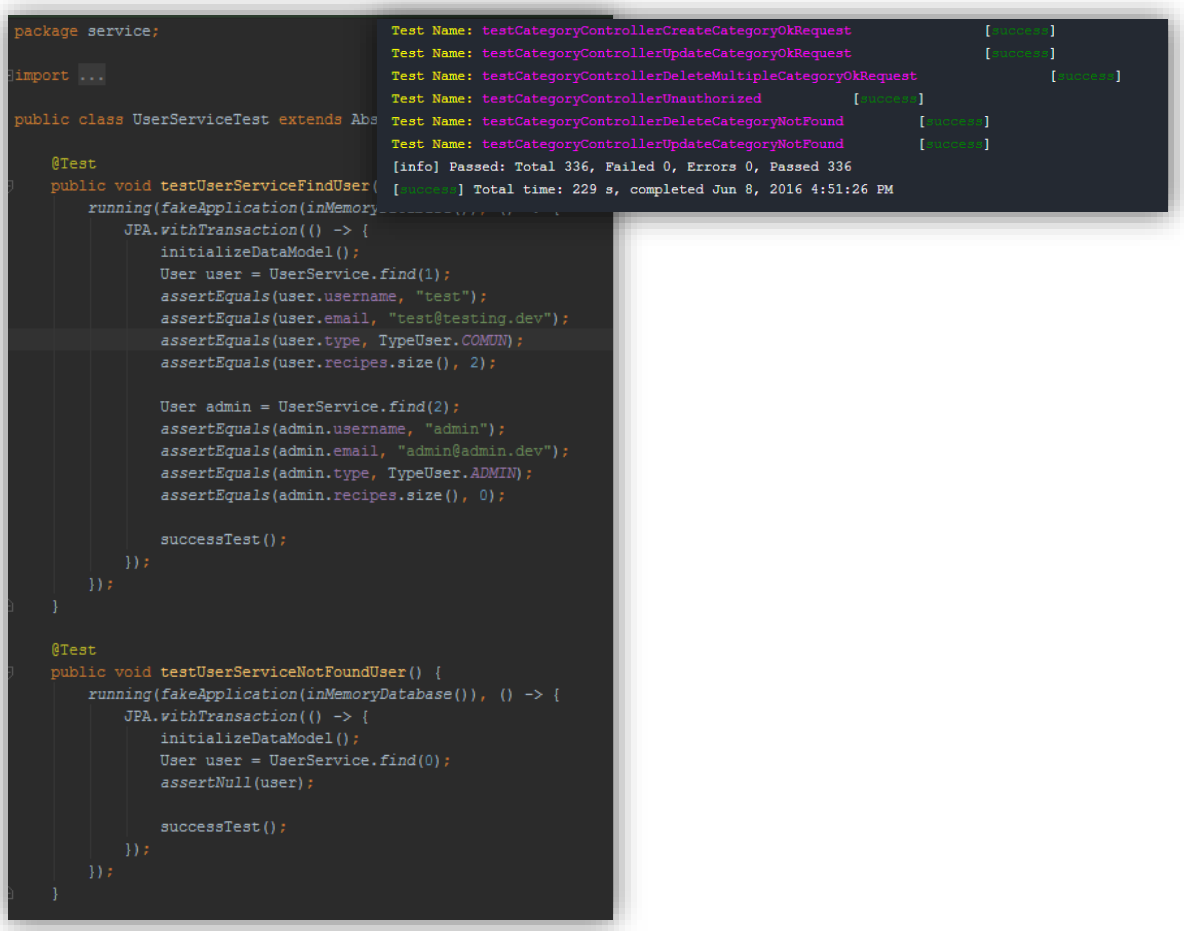
9. TESTING

Una parte fundamental de este y cualquier proyecto son los test o pruebas unitarias. Gracias a esto se puede asegurar el buen funcionamiento de una aplicación sin necesidad de probarla manualmente, aunque un doble chequeo es recomendable siempre.

Como el proyecto se basa en dos aplicaciones con lenguajes distintos, se ha necesitados hacer una batería de test para cada aplicación en su lenguaje.

9.1. Play y JUnit

Para la API REST, escrita en Java, se ha usado las clases y métodos que proporciona Play para el testing junto a los chequeos de JUnit. Esto ha permitido generar una gran cantidad de test para asegurar el buen funcionamiento de la API, que es la base de todo el proyecto. Para mostrar el resultado de los test poco a poco se ha tenido que usar métodos de Java, ya que este conjunto de librerías solo muestra el resultado final o los errores.



9.2. Angular Mocks y KarmaJS

En la parte del front-end también se han añadido test, usando Angular Mocks y KarmaJS. Esto permite crear mockups de las llamadas a la API, del $rootScope básico u otros parámetros que se necesitan previamente.

```
describe('Controller RecipeAll', function() {
  beforeEach(inject(function($rootScope, $controller) {
    $scope = $rootScope.$new();
    ctrl = $controller('RecipeAll', {
      $scope: $scope
    });
    $httpBackend.when('HEAD', /http:\/\/localhost:9000\/recipes\/(.*)/).respond(200, {});
    $httpBackend.when('GET', /http:\/\/localhost:9000\/recipes\?(.*)/).respond(200, {});
    $httpBackend.when('GET', /views\/(.*)/).respond(200, {});
    $httpBackend.when('POST', /http:\/\/localhost:9000\/auth\/check(.*)/).respond(200, {
  }));

  afterEach(function() {
    $httpBackend.verifyNoOutstandingExpectation();
    $httpBackend.verifyNoOutstandingRequest();
  });

  it('initialize controller', function() {
    $httpBackend.expectPOST('http://localhost:9000/auth/check');
    $httpBackend.expectGET(/http:\/\/localhost:9000\/recipes(.*)/).respond({data: []});
    $httpBackend.flush();
    expect($scope.recipes).toEqual([]);
  });
```

```
  - Controller RecipeCreate :
    * initialize controller : ok
    * difficulty method : ok
    * visibility class method : ok
    * visibility icon method : ok
    * save method : ok
    * save error method : ok
    * publish method : ok
    * publish error method : ok
  - Module RecipeFilters :
    - Filter capitalize :
      * capitalize the first letter of sentence w
      * capitalize the first letter of sentence w
    - Filter humanized :
      * humanize existing difficulty : ok
      * humanize no existing difficulty : ok
    - Filter duration :
      * duration is a date : ok
      * duration isn't a date : ok

Browser results:

  - Chromium 37.0.2062 (Ubuntu 0.0.0): 45 tests
    - 45 ok
```

9.3. Pruebas de usabilidad

Aparte de las pruebas unitarias, la aplicación ha sido probada por usuarios reales mediante la plataforma heroku. Esto permite que los usuarios reporten fallos y mejoras, las cuales se discuten en la reunión de Scrum.

10. CÓDIGO E INSTALACIÓN

Todo el código de la aplicación se encuentra en GitHub, es los siguientes repositorios:

- API: https://github.com/JoseVte/tfg-recetarium
- Front-end: https://github.com/JoseVte/tfg-recetarium-angularjs

Para empezar la instalación, lo primero es tener instalado las herramientas necesarias:

- Java (versión >= 8) para la API
- Node (versión >= 0.12) para el front-end

Lo siguiente será descargar la API, ya sea clonando el repositorio o descargándolo en un archivo ZIP. Una vez descargado, en un terminal accedemos a la carpeta donde se encuentre e iniciamos la API:

```
activator run
```

Por defecto la API usará el puerto 9000. Play descargara automáticamente todas las librerías necesarias para el funcionamiento. Para terminar la instalación se necesitara añadir las siguientes variables de entorno:

Variable	Descripción
DATABASE_RECETARIUM_URL	Url completa de la base de datos: mysql://user:pass@localhost/db_name?reconnect=true
FRONT_END_URL	Url para el front-end: http://localhost:8000
PUSHER_APP_ID	Datos necesarios para el funcionamiento de pusher, para obtenerlo necesitamos crear una aplicación nueva: https://pusher.com
PUSHER_KEY	
PUSHER_SECRET	
SECRET_PLAY	Clave usada para la seguridad de la aplicación
SMTP_SERVER	Servidor para el sistema de emails
SMTP_PORT	Puerto para el servidor
SMTP_USER	Usuario para el servidor
SMTP_PASS	Contraseña para el servidor

Para probar que la API esté instalada correctamente se puede acceder a http://localhost:9000, donde se podrá ver una página con el listado de endpoints.

Si todo funciona correctamente, procedemos a la instalación del front-end. Al igual que con la API, entramos en la carpeta y ejecutamos el siguiente comando:

```
node web.js
```

El front-end no requiere ninguna variable de entorno para funcionar. Por defecto usa el puerto 8000. La comunicación se realiza mediante una configuración añadida en el app.js, la cual busca a la API en el puerto 9000.

Para probar que funciones se puede acceder a http://localhost:8000

Con esto ya tendremos nuestra aplicación instalada y desplegada.

11. CONCLUSIONES

Tras acabar el desarrollo del proyecto, se puede concluir que se han cumplido todos los objetivos propuestos inicialmente y los surgidos durante el desarrollo.

Se ha logrado la implementación de una API REST totalmente funcional, estable, segura y bien documentada, que se puede adaptar a cualquier front-end, independientemente del lenguaje. Dicha API REST no solo permite la gestión de recetas por parte de los usuarios, sino que además pueden interactuar con otros usuarios y encontrar nuevas amistades.

Para la parte de interfaz del usuario se ha implementado una web en AngularJS que usa la API anteriormente descrita, facilitando el uso de la aplicación en lo máximo posible.

Todos los mensajes y textos del sistema tienen traducción entre inglés y español, por lo que la aplicación puede llegar a ser usada por gente de distintos países a España.

Se ha hecho uso de los conocimientos aprendidos durante la carrera, en la especialidad de Ingeniería de Software, y más concretamente en las asignaturas del último año como *Aplicaciones distribuidas en Internet*, *Seguridad en el diseño software*, *Ingeniería web*, *Metodologías ágiles de diseño software*.

Durante el transcurso de las asignaturas he ido asimilando pautas, patrones y consejos que han sido aplicados en el desarrollo de este proyecto, como por ejemplo el uso de JWT para la autentificación o el patrón DAO + Services para el acceso a los datos.

Para el desarrollo del proyecto se han seguido una serie de pautas, gestionando el tiempo disponible para poder trabajar en las tareas del sprint y lograr los objetivos establecidos a corto plazo. Sin Trello esta tarea de gestión sería imposible de realizar.

La elección de Play Framework como base de la API fue discutida en la primera reunión de Scrum, y al final a resultado un acierto, gracias a la facilidad de implementación y los conocimientos previos que adquirí en la asignatura *Metodologías ágiles de diseño software*, donde ya se usó previamente.

La elección de AngularJS como framework de front-end también fue discutida en dicha reunión, pero en este caso no ha habido conocimiento previo, por lo que el desarrollo fue más costoso a nivel de aprendizaje, pero igual de fructífero. Ya que AngularJS, al igual que Play, permite ampliar sus funcionalidades mediante módulos, a nivel de desarrollo ha costado más integrar distintos módulos para que operen que tener que desarrollar nuevas funcionalidades.

Ya que la informática nunca tiene límites, el proyecto siempre se puede mejorar y añadir nuevas funcionalidades. Algunas de ellas que han sido planteadas pero no desarrolladas por falta de tiempo son:

- Aplicación front-end para Android/IOS
- Calificar las recetas en un Top 10/25/50
- Aumentar la zona de administración con nuevas opciones

- Posibilidad de crear libros de recetas agrupando varias
- Compartir una receta mediante las redes sociales
- Traducción a mas lenguajes

12. REFERENCIAS

Ambas aplicaciones se han desplegado en internet usando Heroku como sistema y se puede acceder a ellas mediante los siguientes links:

- API: https://recetarium.herokuapp.com
- Front-end: https://recetarium-angular.herokuapp.com

El código de las aplicaciones se encuentra en estos repositorios de GitHub:

- API: https://github.com/JoseVte/tfg-recetarium
- Front-end: https://github.com/JoseVte/tfg-recetarium-angularjs

A continuación aparece una lista con todos los sistemas y librerías usados durante el desarrollo:

- GitHub: https://github.com
- Trello: https://trello.com
- Slack: https://slack.com
- Heroku: https://www.heroku.com
- Atom: https://atom.io
- Intellij IDEA: https://www.jetbrains.com/idea
- Jenkins: https://jenkins.io
- Travis: https://travis-ci.org
- Wercker: http://devcenter.wercker.com/learn/basics/introduction.html
- Mailtrap: https://mailtrap.io
- Heroku: https://www.heroku.com
- JPA y Mysql: https://docs.jboss.org/hibernate/entitymanager/3.6/reference/en/html_single
- Bower: https://bower.io
- NPM: https://www.npmjs.com
- Gulp: http://gulpjs.com
- AngularJs: https://angularjs.org
- Modulos de AngularJS: http://ngmodules.org
    - Angular Material: https://material.angularjs.org/latest
    - Angular Translate: https://angular-translate.github.io
    - textAngular: http://textangular.com
    - Angular infinite-scroll: https://sroze.github.io/ngInfiniteScroll
    - Angular mocks: https://docs.angularjs.org/api/ngMock
    - Angular datatable: https://github.com/daniel-nagy/md-data-table
- PlayFramework: https://www.playframework.com/documentation/2.4.x/JavaHome
- Librerías de Java/Play:
    - jose4j (JWT): https://bitbucket.org/b_c/jose4j/wiki/Home
- DropZone: http://www.dropzonejs.com
- KarmaJS: https://karma-runner.github.io/0.13/index.html

- MomentJS: http://momentjs.com
- Google Material icons: https://design.google.com/icons
- Gravatar: https://en.gravatar.com
- Pusher: https://pusher.com